FLAIRS-90
3rd Florida Artificial Intelligence Research Symposium

Welcome to the THIRD annual meeting of FLAIRS! IF (meetingcount > 2) THEN the_society_is_thriving (cf .99)!

For those of you attending for the first time, FLAIRS was organized in 1987 by a group of AI researchers and professionals in academe and in industry, with the intent of establishing a permanent network of activity for the Florida AI community. The overwhelming success of the '88 and '89 symposia (exceeding even our modest expectation that we would entirely restructure the universe and life as we knew it) attests to the great vitality of that community, and we hope this year to stimulate an even higher level of interest. Also, this year, the participation in FLAIRS has taken on a more conspicuously global character, with papers in this volume reflecting research by authors from eleven states, Canada, England, Germany and Australia.

The theme of FLAIRS-90 is knowledge engineering, and among the AI-knowledgeable whose presence we're especially happy to have engineered are three nationally prominent researchers: Patrick Hayes, James Bezdek and Eugene Charniak, who will speak, respectively, on common sense, neural nets and language comprehension. We hope you will exhibit the common sense to bring your neural hardware to each of their speeches, to engage in some interesting language comprehension.

Finally, on this third year of FLAIRS, we are happy to present: sixty-four papers, eight concurrent sessions, seven planned tutorials, three keynote speakers, many exhibits, and a banquet with pretty good food. (As well as a post-proceedings volume, Advances in Artificial Intelligence Research, now available for the '88 conference from JAI Press!) For a well-stocked exhibit hall, we hasten also to proclaim our appreciation to the very supportive sponsors and exhibitors listed on the next page.

Thanks, finally, to our tireless Organizing Committee - the people who multitasked valiantly against the onslaught of time, the non-onslaught of money, and the generalized encroachments of entropy in all of its guises. And special thanks to you, the authors and attendees, whose participation has been the essential element to bring the intelligence of AI in Florida to the attention of AI's everywhere.

Mark Fishman
Proceedings Chair
for the
FLAIRS Organizing Committee 10
The FLAIRS Organizing Committee

General Chair: Avelino Gonzalez - UCF
Program and Promotion Chair: Douglas Dankel II - UF
Speaker Arrangements Chair: Rita Rodriguez - UWF
Proceedings Chair: Mark Fishman - Eckerd College
Financial & Exhibits Chair: Don Thomas, Cognitive Engineering, Inc.
Tutorial Co-Chair: Dave Lawson - Stetson University
Tutorial Co-Chair: Alex Pelin - FIU
Industrial Liaison Chair: Phil Koltun - Harris Corporation
Panel Arrangements Chair: Fred Hoffman - FAU
Awards Chair: Carlos Segami - UCF
Administrative Chair: Joanne East - UF

Abstracts of Papers were carefully reviewed for presentation at FLAIRS-90 and inclusion in this set of proceedings. We thank our referees:

James Bezdek - UWF
Elizabeth M. Cholawsky - AI Corporation
Donald Crouch - University of Minnesota, Duluth
Douglas D. Dankel, II - UF
Mark B. Fishman - Eckerd College
Kenneth M. Ford - UWF
Larry Hall - USF
Pat Hayes - Xerox PARC
Robert R. Hoffman - Adelphi University
Brian Gaines - University of Calgary
Robert Morris - FIT
John Roach - Virginia Polytechnic Institute
Dan Schwartz - FSU
Carlos Segami - UCF
Christopher R. Westphal - Institute for Defense Analyses
Special thanks to our sponsors and exhibitors

Conference Sponsors

Eckerd College
Florida High Technology and Industrial Council
Florida Institute of Technology
Symbolics, Inc.
University of Central Florida
University of Florida

Exhibitors

Addison-Wesley Publishing
Gold Hill Computers
IBM Information Systems Division
Intellicorp
Inference Corporation
Micro Devices
Sun Micro Systems
Symbolics, Inc.
College of Business, University of Central Florida

...and to all of the following:

Joanne East, for great feats of conference coordination and administrative legerdemain.

Carol Schlabig, for numerous and diverse secretarial contributions.

Roberta Dimouro, for fielding innumerable calls from authors, and stoically enduring the overhead of FLAIRS, while serving as secretary to a collegium of twenty faculty members. (And for pasting on page numbers!)

Alice Willis and Je-Nien Ernst, Eckerd students who partook of the overall production-related scutwork.

Lloyd Chapin of Eckerd College (because he’s the editor’s Dean of Faculty, and encouraged me to spend time on all of this).

...and to all the other deans, department chairs and corporate managers who allowed the steering committee members, forsaking much else, to devote their time and effort to the birthing of this conference.
Table of Contents

Wednesday 8:30 – 10:40 a.m.
Session A - Neural Networks

Incorporating Concurrent Processes into the Object-Oriented Simulation of Neural Networks.........................1
G. Heileman, H. R. Myler, and M. Georgiopoulos
University of Central Florida

A Neural Network Based Solution for Partial Differential Equations.............................................6
P. S. Raju and K. V. Wong
University of Miami

Surface Modeling for Nonrigid Motion Analysis.....................................................10
University of South Florida

Wednesday, 8:30 – 10:40 a.m.
Session B - Reasoning and Search

The Heuristic Control of Logic Problems.................................................................13
T. Colburn
University of Minnesota at Duluth
J. Carciofini
Honeywell Systems and Research Center

Searching with Constraints: A Scheduling Application...........................................19
R. B. Pollack
Florida Institute of Technology
E. A. Pierce
NCR Corporation World Headquarters

A Parallel Knowledge-Base Search Strategy.......................................................23
P. O. Bobbie
University of West Florida

Learning by Analogy on the Connection Machine.................................................28
D. J. Cook
University of Illinois

Knowledge Engineering a Parallel Forward Chaining Inferencing System.........................33
C. P. Industrious, L. O. Hall, and L. Kim
University of South Florida
Wednesday, 3:20 - 5:30 p.m.
Session A - Knowledge Acquisition

Adversary Modeling with Multiple Knowledge Sources......................38
S. Walczak
University of Florida

Techniques for Capturing Expert Knowledge:
An Expert Systems/Hypertext Approach.................................42
L. Lafferty
ENSCO Incorporated
A. M. Koller
Kennedy Space Center
G. Taylor, R. Schumann, and R. Evans
ENSCO Incorporated

Human Engineering the
Knowledge Acquisition Interface: The Evolution of Vista..............49
R. Ahlers
Naval Training Systems Center
M. Schnitzius
Oak Ridge Associated Universities

A Comparative Analysis of Three
Techniques of Knowledge Acquisition for Expert
Systems Applied to the Domain of Literary Preference.................53
J. M. Ernst
Eckerd College

The Cognitive Psychology of Expertise: A Review of the Literature...57
R. Hoffman
Adelphi University

Wednesday, 3:20 - 5:30 p.m.
Session B - Vision

Towards Computing the Aspect Graph
of Constructive Solid Geometry Objects...............................60
R. Bolling and K. W. Bowyer
University of South Florida

Using Saccadic Eye Movements to Create a 2.5-D Sketch.................65
K. M. Simpson and K. W. Bowyer
University of South Florida

Representing the Visual Potential
of a Non-Rigid Assembly of Rigid Parts................................70
M. Sallam and K. W. Bowyer
University of South Florida
Stereo Processing with Artificial Neural Networks.............75
T. L. Clarke
University of Central Florida

Estimating Motion Parameter in Linear Conformal Motion...........80
S. K. Mishra and D. B. Goldgof
University of South Florida

Towards a Representation of Facial Expressions for Recognition and Display.................................85
C. Kambhamettu, D. B. Goldgof, and A. Tokuta
University of South Florida

Thursday, 8:30 - 10:10 a.m.
Session A – Knowledge Acquisition

ICONKAT: Integrated Constructivist Knowledge Acquisition Tool........90
H. Stahl
Cornell University
K. M. Ford
University of West Florida
J. R. Adams-Webber
Brock University
J. Novak
Cornell University

An Object-Oriented Knowledge Acquisition Tool..................96
P. Biswas
Florida State University

KASH: A General Purpose Knowledge Acquisition Tool..............101
C. R. Westphal
Institute for Defense Analysis
D. Tran

A Graphically Oriented Automated Knowledge Acquisition Tool........107
J. M. Sargeant and J. M. Schuerger
Harris Corporation

Thursday, 8:30 - 10:10 a.m.
Session B – Knowledge Representation

Knowledge Representation in Expert Systems Using Reduced Matrices Techniques.........................112
H. Lim and M. Schneider
Florida Institute of Technology

Weak Paraconsistent Logics for Knowledge and Belief..............117
R. Girle
Australia National University
Knowledge Representation in Fuzzy Relational Databases.............122
R. Joshi and M. Schneider
Florida Institute of Technology

Reasoning with Unions of Convex Time Intervals......................127
R. A. Morris and L. Al-Khatib
Florida Institute of Technology

Thursday, 8:30 - 10:10 a.m.
Session C - Natural Language

Analysis of General Tabular
Parsing for Natural Language Translation..............................133
F. Hadlock
Tennessee Technical University
M. Fishman
Eckerd College

Use of Fuzzy Relational Information
Retrieval Techniques for Generating Control
Strategies in Resolution-Based Automated Reasoning..............140
L. J. Kohout and Y. G. Kim
Florida State University

LR/1, A Prolog-Like System for
Reasoning with Imprecise Linguistic Information..................145
H.-T. Chung and D.G. Schwartz
Florida State University

Thursday, 1:30 - 3:10 p.m.
Session A - Learning

Performance-Driven Knowledge Transformation......................149
L. B. Holder
University of Illinois

A Formal Constructivist Model Of Knowledge Revision..............154
R. Yager
Iona College
K. Ford
University of West Florida

Experiments in Routing an Autonomous Land
Vehicle with a Weakly Inductive Learning Algorithm...............159
M. D. Petty, T. J. Frederick and J. M. Moschell
University of Central Florida
Evaluation of the Learning Process
in a Self-Adapative Expert System............................164
D. R. Berlin and M. Schneider
Florida Institute of Technology

Thursday, 1:30 - 3:10 p.m.
Session B - Knowledge Representation

Artificial Networks for Qualitative Reasoning in Design..........168
J. H. Garcelon and G. E. Nevill
University of Florida

Frame Planner.......................................................173
F. M. Brown, S. S. Hundal and C. L. Araya
University of Kansas

How Fuzzy Should a Neuron Be?.................................177
J. Barone
Loki Software, Inc.

Thursday, 3:30 - 5:10 p.m.
Session A - System Architecture

Constraint Management for Integrated Systems......................184
E. E. Silverstein and P. Sun
McDonnell Douglas Corporation

A Model Based Power System Controller for the Space Station.....189
R. Morris
Florida Institute of Technology
A. J. Gonzalez
University of Central Florida

A Knowledge-Based Writer's Aid for Simplified English............194
I. Syu, S. D. Lang, P. J. Kincaid and M. Thomas
University of Central Florida

Thursday, 3:30 - 5:10 p.m.
Session B - Verification and Validation

Validation of Expert Systems......................................197
S. Smith
East Tennessee State University
A. Kandel
Florida State University
Expert System Validation as It Applies to Expert Systems Using Frame-Based Knowledge Representation..................202
A. M. Cheng
Honeywell Military Avionics Division
L. O. Hall
University of South Florida

Thursday, 3:30 - 5:10 p.m.
Session C - Knowledge Representation

Automatic Programming for Array Manipulation.........................207
J. Wu
Florida Atlantic University

Automatic Generating of Interesting Hypothesis Concerning Oncological Medical Data.........................210
L. J. Kohout
Florida State University
I. Stabile
University of Florida

J. J. Kelly
Martin Marietta
K. M. Ford
University of West Florida

Knowledge Engineering in LILOG...........................................220
G. Klose and K. v. Luck
IBM Germany

Friday, 1:30 - 3:30 p.m.
Session A - System Architecture

Distributed Object-Oriented Inferencing in Ada.....................225
R. Simonian
Harris Space Systems Corporation

An Overview of XX: An Assistance System for Hydrocarbon Exploration........................................230
J. C. Bezdek
University of West Florida
G. Biswas
Vanderbilt University
R. Cannon and C. Kendall
University of South Carolina

Towards a More General Architecture for Intelligent Tutoring Systems........................................240
H. W. Kegeleman, L. W. Hawkes and S. J. Derry
Florida State University

xii
An Architecture for Ruled-Based
Knowledge Representation and Parallel Inferencing..................245
D. E. Tamir, R. A. Morris and M. Schneider
Florida Institute of Technology

A Knowledge-Based System for Transporting Software..................250
S. Leong and F. Y. Wu
University of Miami

Friday, 1:30 - 3:30 p.m.
Session B - Knowledge Representation

HALO -- A Fuzzy Programming Language.................................254
D. Clark and A. Randal
Florida State University

Constraint-Based Modeling of Behaviors.................................258
J. Burg, C. Hughes and J. M. Mosherl
University of Central Florida

A Neural Network Approach to the
Determination of Aquifer Parameters.................................263
A. R. A. Aziz and K. V. Wong
University of Miami

Distributed Knowledge Bases Via Parallel Logic Programming.........268
S. C. Greer, R. M. Butler, S. R. Wallace and J. L. Solano
University of North Florida

Friday, 1:30 - 3:30 p.m.
Session C - Neural Networks

Neural Network Simulation on a Local Area Network..................271
H. K. Brown and D. Cross
University of Central Florida

A Discrete Temporal Model of Dynamic
Processes in the Dendritic Trees of Neurons.........................274
R. Gawronski, F. Anger and R. Rodriguez
University of West Florida

Neural Bit-Slice Computing Element.................................280
J. Yestrebsky, P. Basehore and J. Reed
Micro Devices

Neural Network Learning Using a Constrained Weight Space Search....284
M. Georgiopoulos and G. H. Heileman
University of Central Florida

A Dynamic Gigaconnect Neural Network Architecture..................289
H. Brown and D. F. Lange
University of Central Florida
Late-Breaking Papers

On the Application of Functional Programming to Natural Language Processing..........................292
S. J. Murrell and R. T. Plant
University of Miami

Validation Tools for Expert Systems.................................295
U. G. Gupta and J. Biegel
University of Central Florida

A Biologically Based Neural Network Model.........................300
G. Cornell
University of Central Florida

Limitations of the Applications of Outstar Learning Theory........305
to both Analog and Digital Implementation Media
E. Nold
Martin Marietta
L. Canney
Martin Marietta
M. Georgiopoulos
University of Central Florida

List of Authors..............................................................311
Index.............................................................313
Incorporating Concurrent Processes into the Object-Oriented Simulation of Neural Networks *

Gregory L. Heileman  Harley R. Myler  Michael Georgioupolos

Department of Computer Engineering
Department of Electrical Engineering
University of Central Florida, Orlando, FL 32816

Abstract

An object-oriented approach to the simulation of artificial neural networks allows the design of specialized models to proceed from a higher level of abstraction than is possible in simulation systems using a procedure-oriented methodology. However, this approach does not address the critical issue of the time required to perform the simulation. The load that a neural network simulation places on conventional computer systems is quite heavy. Simulating even a relatively small neural network is a computationally intensive task that requires huge amounts of computer resources. In this paper, the issues of implementing an object-oriented neural network simulations on parallel computer architectures are addressed. This is accomplished through the use of the parallel programming language Concurrent C++. This language utilizes object-oriented programming capabilities while providing the ability to specify parallel execution. Such an approach allows the simulation of large neural network models on parallel computers.

1 Introduction

The tremendous interest that has recently surfaced regarding artificial neural network computing models has led researchers to propose the use of such models in many areas of artificial intelligence research. One approach used to

*This research was supported by a grant from the Florida High Technology and Industry Council with matching support from Martin Marietta Electronic Systems Division.

Proceedings of the 3rd Florida Artificial Intelligence Research Symposium, 1990, by the Florida AI Research Society

c. 1990 FLAIRS 0-9620-1732-9/90/0350-0001

implement nontrivial neural network makes use of object-oriented programming techniques [5]. The data abstraction facilities offered by an object-oriented approach provide a number of benefits. First, the manner in which data is encapsulated allows low-level implementation details to be effectively hidden from the user, who must access the facilities provided by the simulation system at a higher-level of abstraction. In addition, the ability to inherit the functionality of a generic neural network model circumvents the need for the user to write low-level routines that directly manipulate the computer representations of the network elements. This leads to a highly flexible simulation environment in which one can easily extend or modify the functionality of specific neural network models. Therefore, the development of object-oriented neural network simulation systems offers researchers the capability of easily simulating novel models adapted to their specific application. Furthermore, this capability does not require researchers to understand the complex issues involved in implementing the low-level routines that directly manipulate the network elements — they must only understand the functionality offered by the high-level routines that access these network elements.

Although an object-oriented approach allows the design of specialized neural network models to proceed from a higher level of abstraction than is possible in simulation systems using a procedure-oriented methodology, it does not address the critical issue of the time required to execute neural network simulations. The load that such simulations place on conventional computer systems is quite heavy. Simulating even a relatively small neural network is a computationally intensive task that requires huge amounts of computer resources. This results from the fact that such computer systems are unable to directly implement the parallel nature of neural network models; instead, they are simulated sequentially, computing the
output of each node in the network in turn. The inefficiency of such
implementations tends to restrict the size of the neural network simula-
tion to a few hundred nodes. However, it is widely held that the full potential of neu-
ral network models cannot be fully exploited without the use of much larger networks, and therefore the restrictions placed on network size by current simulation systems is considered severe.

In this paper, we address the issue of implementing an object-oriented neural network simulation on parallel computer architectures. This will be accomplished through the use of a newly developed programming language, Concurrent C++, that utilizes object-oriented programming capabilities while providing the ability to specify parallel execution [3,4]. Such an approach will allow the simulation of large neural network models on parallel computers.

2 A Generic Neural Network Model

An artificial neural network can be considered a directed graph in which the edges of the graph represent network weights (or synapses), and the nodes of the graph represent the computing elements (or neurons). Each node in this model has an activation value \( a_i(k) \) which is a function of the inputs to the \( i \)-th node after \( k \) iterations; this value is passed through a function \( f \) to determine the node output \( o_i(k) \). Next, the output value is propagated through the network to other nodes in the system. The synaptic weight \( w_{ij} \) associated with each edge in the graph determines the amount of effect the \( i \)-th node has on the \( j \)-th node.

All of the inputs to node \( j \) at time \( k \) are combined according to some operator, along with an internal offset \( \theta_j \), to produce the net input to node \( j \), \( u_j \). One of the most common approaches is to simply sum the inputs to the \( j \)-th node, this is expressed as the following propagation rule:

\[
    u_j(k) = \sum_i a_i(k) w_{ij} + \theta_j
\]

(1)

In this example, a positive weight value can be considered an excitatory connection and a negative weight value can be considered an inhibitory connection.

The activation level of node \( j \) at time \( (k + 1) \) is a linear or nonlinear function of some or all of the following components: the current net input \( u_j(k) \), the current activation value \( a_j(k) \), and the previous net inputs. Thus, the new activation value \( a_j(k+1) \) is given by the following activation function \( F \):

\[
    a_j(k+1) = F(a_j(k), u_j(k), u_j(k-1), ...)
\]

(2)

It is the pattern of activation levels in the nodes that determines what the system is representing at any given time. Thus, it is useful to view the processing in these systems as an evolution, through time, of a pattern of activity in the network.

All that remains to completely specify a neural network model is to determine a learning rule and network topology. The learning rule determines the method in which the connection weights between nodes are adjusted in order to improve performance. A major portion of neural network research is directed at finding better learning algorithms.

3 Object-Oriented Implementation

Object-oriented programming languages center around the use of classes to provide data abstraction facilities for user-defined data types. The class represents a means of directly implementing abstract data types. That is, a class is capable of representing the data elements of a user-defined data type, and is also responsible for specifying and controlling the access to these data elements. In the C++ programming language, class declarations consist of two parts: a specification and a body. The class specification provides a "user-interface" for other classes requiring the use of the class. This interface is simply a list of the accessing routines available for manipulating the data elements in the class. In the object-oriented terminology, these accessing routines are called methods. The class body contains the bodies of the methods declared in the class specification. An excellent treatment of the general principles involved in the object-oriented design of software systems is provided in [7]. For detailed discussions of C++ see [2] and [8].

The object-oriented implementation of the generic neural network model presented in Section 2 involves the creation of two classes: one class to represent the nodes in the network, and another class that uses the node class to represent the network itself. A simplification of the specification for the class used to implement the network nodes is shown in Figure 1. The section of the node class specification marked as private cannot be directly accessed by other classes; instead, these classes are required to access the private data elements via the methods provided in the public section of the class specification. For
class node_class {
  private:
  int num_inputs, num_outputs;
  float net_input, output;
  input_edge *input_connection;
  output_edge *output_connection;
  public:
  node(int NumInputs, int NumOutputs, input_edge *InputConnection, output_edge *OutputConnection); // constructor
  node(); // array constructor
  "node"(); // destructor
  int NumInputs(); // returns # inputs
  int NumOutputs(); // returns # outputs
  float NetInput(); // returns net input
  float Output(); // returns output
  float Input(int InputNum); // returns an input
  float Weight(int InputNum); // returns a weight
  virtual void ComputeNetInput();
  virtual void ComputeOutput();
};

Figure 1: C++ class specification for node_class.

class ann {
  private:
  int num_nodes, num_layers;
  int *nodes_per_layer; // # nodes in a layer
  int *layer_node; // node # of a specific
                     // node in a given layer
  node_class *node; // list of network nodes
  public:
  ann(ann_specs *Specs); // constructor
  ann(ann #); // copy initializer
  "ann"(); // destructor
  int NumberOfLayers();
  int NumberOfNodes(int Layer);
  float NetInput(int Layer, int Node);
  float Output(int Layer, int Node);
  float *Weights(int Layer, int Node);
};

Figure 2: C++ class specification for class ann.

4 Parallel Implementation

In the following sections we discuss how the incorporation of concurrent processes into the classes discussed in Section 3 can be accomplished. The resulting neural network class will make use of both the concurrent processing and data abstraction capabilities offered by the Concurrent C++ programming language. The data abstraction facilities of an object-oriented language will be used to provide a high-level, robust interface for other classes requiring the use of the neural network class. The concurrent programming facilities will allow this class to efficiently execute on parallel computing hardware thereby exploiting the inherent parallelism of neural network models.

4.1 Concurrent Programming Model

Concurrent programming languages are based on the use of processes. A process has its own thread of execution, stack, and machine registers. Thus, two processes may execute simultaneously on separate processors, or they may be time-sliced on a single processor. The concurrent programming facilities provided by Concurrent C++ are extensions of the Communicating Sequential Processes [6]
and Distributed Processes [1] concurrent programming models. These models also form the basis for the concurrent programming facilities offered by the Ada programming language.

Concurrent C++ processes communicate using message passing. These messages can be either synchronous transactions or asynchronous transactions. In the synchronous case, a client process sends a message to a server process requesting it to perform some service. The client process then must wait for the server process to accept the message and perform the desired service. Upon completion, the server process may return results to the client process which is then free to resume execution. In the asynchronous case, the client process does not wait for the server process to receive the message, and it is not possible for the server process to return results to the client process. Which type of message passing transaction is most efficient depends upon the intended application. There are situations where the synchronous approach is more appropriate and vice versa. However, programs that use synchronous transactions are generally easier to understand as well as debug.

4.2 A Concurrent Base Class

In the concurrent implementation of the neural network class, a process will be associated with each node in the network. Ideally this will maximize speedup allowing each node to execute on a separate processor. If this is not the case, a number of nodes will be required to execute on a given processor. In either situation, each node in the network will receive its input from, and pass its output to, other nodes in the network using message passing. Each network node will be allowed to compute its net input and output functions, and pass its output value to other nodes whenever transactions have been received that request these operations.

One approach that can be used involves initiating a separate synchronous transaction between a node process and every node that receives its output whenever the node computes its output function. The specification and body of a process capable of executing these tasks is shown in Figure 3. The process specification declares that nodeProc can act as a server for a client process requesting the synchronous transactions Input or Output. The Input transaction is used to assign a value to a specific input of the server node. The Output transaction directs the server

```
process spec nodeProc {
  trans void Input (int ConnectionNum, float Val);
  trans void Output();
};

process body nodeProc() {
  int i;
  for (; ; ) {
    select {
      accept Input(x, y);
      input_connection[x].input = y;
    or
      accept Output();
      ComputeNetInput();
      ComputeOutput();
      for (i=0; i<num_outputs; i++)
        output_connection[i].node_id.Input(output);
    }
  }
}
```

Figure 3: Concurrent C++ process specification and body for nodeProc.
	node to first compute its net input and output values using the ComputeNetInput and ComputeOutput methods provided by the node class, and then pass this output value on to all nodes connected to the server node. The output value is passed along to other nodes by initiating synchronous node transactions with those nodes. In this sense, the server nodes becomes a client node.

In the Concurrent C++ implementation of node_class, nodeProc is declared as a private member of the class as shown in Figure 4.

```
class node_class {
  private:
    nodeProc neuron;
    int num_inputs, num_outputs;
    float net_input, output;
    input_edge *input_connection;
    output_edge *output_connection;
  public:
    ...
};
```

Figure 4: Concurrent C++ specification for node_class.
It is interesting to note that the data abstraction facilities of Concurrent C++ allow the C++ interface to the ann class shown in Figure 2 to remain unchanged. Therefore, a user of this class does not have to be concerned with the complex and time consuming implementation details involved in mapping a particular neural network model to a parallel architecture. Instead, the user may build novel neural network simulation models using the same high-level interface provided in the strictly object-oriented approach, with the added benefit of being able to efficiently execute these computationally intensive simulations on parallel computing hardware.

5 Conclusion

Artificial neural network models typically involve large numbers of interconnected computing elements. Therefore, the ability to execute neural network simulations on parallel computers is considered a necessity if non-trivial models are to be investigated. The approach presented here involves the specification of parallelism within the problem using a general-purpose parallel programming language, Concurrent C++. The significance of this approach is that it also incorporates the principles of object-oriented software design. The resulting parallel simulation system can therefore be easily modified to develop and test novel neural network models without having to consider the complex issues involved in mapping a the neural network model to the underlying parallel hardware.

REFERENCES


A NEURAL NETWORK BASED SOLUTION FOR PARTIAL
DIFFERENTIAL EQUATIONS

Srinivasa P. Raju and KAU-FUI V. Wong
Department of Mechanical Engineering
University of Miami
Coral Gables, Fl 33124

ABSTRACT

This paper discusses the application of neural networks to solve problems in numerical analysis. An elliptic Equation is solved for illustration. The advantages of this method are highlighted along with its limitations. The discussion need not be limited to elliptics alone but also to other problems as well.

INTRODUCTION

Neural networks offer important new approaches to problem solving because of their adaptivity and their ability to learn as well as their parallelism. Neural networks make use of computing elements that are simplified versions of neurons, the specialized cells that do information processing in the brain. These units are highly interconnected and the details of the connections are of critical importance to the functioning of the network. Each connection has a strength and it is assumed generally that it has a single value and the connections between the computing elements are unidirectional. Further the networks utilise various learning rules that specify how the strengths are set. Since it is well known that the brain is made up of a large number of neurons, the neural network software is also made up of a large number of artificial neurons. These units are all often arranged in physically parallel arrays and are hence known to compute simultaneously. Association, inferential data storage, concept formation and similar such phenomena of "psychological" nature are areas in which these are very efficient. It is for these reasons that they are also called "connectionist" because the information is stored in the value of the strengths of the connections.

These and other features make them different from an expert system or a procedural program and hence they offer an altogether new and revolutionary approach to the solution of problems one might encounter in the sciences and engineering.

In the present work, an attempt has been made to solve problems in numerical analysis using neural networks. In this particular field any work has been rarely observed. It is for this reason that a partial differential equation (PDE) has been solved for illustration, explaining the case with its advantages and its limitations. The objective was to generate data for training by a higher-level procedural program and use the data generated to find a general solution based on surface-fitting and extrapolation. This process was implemented on a personal computer using a commercially available neural network software.

ANALYSIS

Consider, for instance, the Laplace Equation,

\[ \nabla^2 u = 0 \text{ in } \Omega \]  (1)

where \( u \) indicates the exact solution and \( \Omega \) is the domain under consideration. The corresponding boundary conditions may be of the following types:

conditions \( u_0 = U \) on \( \Gamma_1 \)  (2)
conditions \( q_0 = q \) on \( \Gamma_2 \)  (3)

where \( \Gamma_1 \) and \( \Gamma_2 \) are parts of the boundary where one or the other condition applies. The total boundary is \( \Gamma = \Gamma_1 + \Gamma_2 \).

The exact solution \( u_0 \) can be found only for a few simple cases and generally the

Proceedings of the 3rd Florida Artificial Intelligence Research Symposium, 1990, by the Florida AI Research Society

c. 1990 FLAIRS 0-9623-1732-9/90/0350-0006
solution will have to be approximated. This can be done using a set of known non-linearly independent functions \( \phi_1 \) and unknown coefficients \( U_1 \). \( U_1 \) are values of the \( U \) function at certain points or nodes.

Writing the approximation for \( U \) as,

\[
\bar{U} = \sum \phi_1 U_1
\]

and introducing this function into Equation (2) and (3) will produce an error, excepting the case for which Equation (4) is the exact solution i.e., \( U = U_0 \), such that,

\[
\nabla^2 \bar{U} = 0 \quad \text{in} \quad \Omega
\]

and on the boundary,

\[
U_1 \bar{U} = 0 \quad \text{on} \quad \Gamma_1 \quad (6)
\]

\[
\mathbf{q}_1 \cdot \mathbf{n} = 0 \quad \text{on} \quad \Gamma_2 \quad (7)
\]

where

\[
\mathbf{q} = \mathbf{n} \frac{\partial \bar{U}}{\partial \mathbf{n}} \quad (8)
\]

one can now define error function in the domain and on the boundary such that,

\[
\varepsilon = \nabla^2 \bar{U} = 0 \quad \text{in} \quad \Omega
\]

and \( \varepsilon_1 = U_1 \bar{U} = 0 \quad \text{on} \quad \Gamma_1 \quad (9) \)

\[
\varepsilon_2 = \mathbf{q}_1 \cdot \mathbf{n} = 0 \quad \text{on} \quad \Gamma_2 \quad (10)
\]

The aim is now to make this error as small as possible over the domain and on the boundary. In order to do so, the errors can be distributed, and the way in which this distribution is carried out produces different types of weighted residual techniques. The distribution of the error function \( \varepsilon \) can now be carried out by multiplying it by the weighting function \( w \) and integrating over the domain, i.e.,

\[
\int_{\Omega} \varepsilon w d\Omega = \int_{\partial \Omega} \nabla \phi \cdot \mathbf{n} d\Gamma = 0 \quad (12)
\]

Classical finite difference techniques for instance can be interpreted as a special case of Equation (12) for which the weighting functions are the dirac delta functions,

\[
\delta(x - x_i) = \lim_{N \to \infty} \sin \frac{N(x - x_i)}{2} \quad (13)
\]

In the limit this function is zero at every point except where the argument is zero, where it is infinite (i.e., at \( x = x_i \)). Thus it represents a point singularly at the "source" point \( x_i \).

This proves that when one solves for the nodal values \( U_1 \), one obtains the values only at those nodes and the value of the field variable \( U \) at points not lying on the nodes is unknown, or does not have a meaning. Then one goes in for a finer mesh in which the point falls on one of the nodes. But this is a cumbersome and a very time-consuming procedure. The finite element schemes may be used. In the finite element method one assumes the entire domain to be composed of smaller elements and one also assumes that there is a predetermined variation within the element itself of the field variable. These variations are such that the field variables computed at one point are not likely to influence the nodes at the far-off points computationally. This means that the field variable has local dependence and it is assumed that the variable at one point can influence only some of its closest neighbours (this can be proved from the banded structure of the stiffness matrix). Thus it has been found that the finite difference method (FDM) cannot fill it self of point dependence nor the finite element method (FEM) of its local dependence. One can then use neural nets to solve this problem of ascertaining the value of the field variable by a sort of learning-interpolation procedure.

**The Neural Net Approach**

A neural net could be used to obtain the values at the various points. To build and apply one, one has to follow the steps given below.

1. Decide what the network has to learn.
2. Decide on how to represent the information.
3. Generate or obtain data for the network for learning.
4. Decide on the structure of the network (like the number of neurons etc.)
5. Train the network using the data.
6. Test the network for its expertise.
7. Use the trained network to make predictions.

Now for the problem under consideration, the neural net is made to train with the data obtained using the FDM or the FEM and a training fact file is generated which is used to compute the value at any other point. This is more expedient and spares one from increasing the grid size or fitting a surface to obtain the value as in FEM and/or removes the neighbour-influencing-neighbour principle of function of the FEM. An overall surface is fitted with these data to obtain the true value at any point.

**Algorithm**

Consider for instance,

\[
\nabla^2 U = 0 \quad \text{on} \quad 0 < x < 1, \quad 0 < y < 1 \quad (14)
\]

The solution was assumed to be \( U = x^2 - y^2 \). Using this solution the boundary conditions were set. With these boundary conditions and using a finite difference scheme a program was written to generate data for the network. With these data for training, the network was run. The training fact file was later utilized to compute the values at the other points of the region. The results concurred quite well with
the exactly known solution, within the tolerance limits of the neural software used.

PROGRAM IMPLEMENTATION

The procedural program shown in appendix B was written to generate the training fact file. The program utilizes a finite difference scheme to generate facts at select nodes for training. The training fact file contains information on the type of inputs and outputs, the number of neurons to be used, the format and the number of inputs and outputs, and the facts needed for training. Appendix A shows a typical training fact file utilized for this paper. The inputs are the X and Y coordinates and the output is the value of the field variable U corresponding to the inputs X and Y. This fact file was trained on a neural network software to generate a test file. The test file was later utilized to predict the value of the field variable at some select points as mentioned previously.

RESULTS AND CONCLUSIONS

A neural network is certainly one of the effective ways to solve problems in numerical analysis. As regards the limitations and sometimes the unreliability of the results of the neural network, it can be said that if the tolerance limits are reduced, the neural network would be much more effective than the ordinary methods at the expense of more training time. This method certainly brings the predicted values closer to the exact values; however, due to the high tolerance limits or higher training times at low tolerance limits, they are not close to exact. Nonetheless, if both of these problems are solved, neural networks would be unmatched by any other conventional procedure. This approach need not be confined to elliptic equations alone as was done in this paper. It could be used for other types of problems where extrapolation and curve-fitting and/or estimation are involved as in modelling, etc., where this could save time, compared to the exact surface fits, which would involve more work.

REFERENCES


APPENDIX - A

input number 1 2
hidden 5
output number 1 1

display input number 5 5 1 15
display output number 6 5 1 7

MINIMUM
0.00000 0.00000
-1.00000

MAXIMUM
1.00000 1.00000
1.00000

Facts
0.00000 1.00000
-1.00000 0.00000
0.00000 0.50000
-0.25000 0.00000
0.00000 0.00000
0.50000 1.00000
-0.75000
0.50000 0.50000
0.00000 0.00000
0.25000 0.00000
1.00000 1.00000
0.00000
1.00000 0.50000
0.75000
1.00000 0.00000
1.00000

8
APPENDIX - B

10 OPEN "O",#1, "CAT.PRG"
20 DIM A(100,100)
30 INPUT A,B
40 INPUT N,M
50 INPUT KKKK
60 K = 1
70 KK = 1
80 DX = A/(N-1)
90 DY = B/(M-1)
100 PRINT #1,"input number 1 2"
110 PRINT #1,"hidden 5"
120 PRINT #1,"output number ";
N-2:M-2
130 PRINT #1,
140 PRINT #1,
150 PRINT #1,
160 PRINT #1,"display"
input number 5 5 1 15"
170 PRINT #1,"display output"
number 6 5 ";N-2:(N-2)^2 + 5
180 PRINT #1,
190 PRINT #1,
200 PRINT #1,
210 PRINT #1,"MINIMUM"
220 PPPP = DX/2*(KKK-1)
230 QQQQ = DY/2*(KKK-1)
240 PRINT #1,USING"####.###"
;PPPP:QQQQ
250 FOR I = 1 TO N-2
260 FOR J = 1 TO M-2
270 PRINT #1,USING"####.###" ;.05;
280 NEXT J
290 PRINT #1,
300 NEXT I
310 PRINT #1,
320 PRINT #1,
330 PRINT #1,
340 PRINT #1,"MAXIMUM"
350 PRINT #1,USING"####.###" ;DX:DY
360 FOR I = 1 TO N-2
370 FOR J = 1 TO M-2
380 PRINT #1,USING"####.###" ;1.2;
390 NEXT J
400 PRINT #1,
410 NEXT I
420 PRINT #1,
430 PRINT #1,
440 PRINT #1,
450 PRINT #1,"facts"
460 FOR KKK = 1 TO KKKK
470 FOR NNN = 1 TO KKKK
480 FOR I = 1 TO N
490 FOR J = 1 TO M
500 A(I,J) = .01
510 NEXT J
520 NEXT I
530 FOR JJ = 1 TO 100
540 FOR I = 1 TO N
550 FOR J = 1 TO M
560 XY = (A(I+1,J)+A(I-1,J))/
SURFACE MODELING FOR NONRIGID MOTION ANALYSIS

Edward M. Wokabi  Dmitry B. Goldgof
Kevin Bowyer
Department of Computer Science and Engineering
University of South Florida
Tampa, Florida 33620 U.S.A.

ABSTRACT

In this paper we investigate the applicability of superquadric surfaces to modeling of nonrigid objects. In particular, we describe a procedure for fitting superquadric surface model to three-dimensional data of the left ventricle (LV) of the heart, and for evaluation of this model for both static LV surface modeling and, dynamic LV surface modeling, (i.e. left ventricle motion analysis).

In the first section, background is given concerning recent work on nonrigid motion analysis. Then, the general theory behind superquadric modeling is presented along with the formulas and the parameters necessary to determine the shape of a 3-D model. Also, the fitting of 3-D data points to a superquadric model using a nonlinear least square minimization method is described. Finally, the hyperquadric modeling, and the future research is discussed.

INTRODUCTION

The estimation of three dimensional motion parameters for an object is one of the major research topics in computer vision. Recently, a new area of research has emerged: the motion analysis of nonrigid objects. Several researchers have already approached specific problems in this area of nonrigid motion analysis. In particular, (Chen 1985; Chen and Penna 1996) investigated elastic motion of objects under the restricted assumption of isometry. (Koenderink 1984) has studied several similar cases of bending deformations, while (Goldgof, Lee and Huang 1988) have used Gaussian and mean surface curvatures to analyze nonrigid motion.

In all approaches to nonrigid motion estimation, nonrigid surface modeling is an important part of the solution and remains an unsolved problem. In particular, curvature-based algorithms require point correspondences between consecutive time frames. One approach to finding point correspondences is based on considering an analytical model of nonrigid object. Recently, superquadric models have proven flexible enough to represent a wide class of objects, and simple enough to be recovered when applied to 3-D data from rigid objects (Bajcsy and Solina 1987; Boult and Cross 1987; Pentland 1986). This paper investigates the use of superquadrics in modeling nonrigid objects in general, and the LV of the heart in particular. Consequently, we consider the applicability of this model in both the static and dynamic motion analysis of the LV. The successful modeling will allow us to find surface point correspondences of the LV at different time instances during the heart cycle. This is especially important for curvature-based algorithms (Goldgof, Lee and Huang 1988) which require point correspondences between two time frames for local motion recovery.

The importance of the application of nonrigid motion analysis methods to cardiac and other imaging cannot be overstated since it will provide quantitative data for cardiac and other medical imaging such as the lungs. Similarly, these methods can be extended to such diverse areas as human face recognition for high-speed, low-bandwidth teleconferencing, material deformation studies, and clouds tracking for weather predictions.

SUPERQUADRIC MODELING

In this paper, we use superquadrics to model the LV of the heart. Superquadrics are a family of parametric shapes which have been proposed for use as primitives for shape representation in computer vision (Solina and Bajcsy 1990). Superquadrics can be compared to lumps of clay that can be glued together to create very realistic looking models of natural objects (Pentland 1986). Simple superquadric equation in two dimensions

\[
\frac{x^2}{a_x^2} + \frac{y^2}{a_y^2} = 1
\]

is an extension of a quadratic equation

\[
\frac{x^2}{a_x^2} + \frac{z^2}{a_z^2} = 1
\]

with the exponent 2 replaced by \( \frac{1}{2} \).

A 3-D superquadric surface is defined as the closed surface spanned by the the vector

\[
\mathbf{r}(\eta, \omega) = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} a_x \cos^k \eta \cos^m \omega \\ a_y \cos^k \eta \sin^m \omega \\ a_z \sin^k \eta \end{bmatrix}
\]

having \( x, y, \) and \( z \)-components specified as functions of the angles \( \eta \) and \( \omega \) in the intervals \(-\pi/2 \leq \eta \leq \pi/2 \) and \(-\pi \leq \omega \leq \pi \) (Boult and Cross 1987).
The parameters \(a_x, a_y, a_z\) define the superquadrics size along the \(x\), \(y\) and \(z\) axes respectively, whereas, the parameters \(\varepsilon_1\) and \(\varepsilon_2\) define the relative shape of the superquadric in the \(xz\) and \(yz\) directions. By varying the these parameters, the superquadrics can model a large set of standard building blocks (Boult and Gross 1987).

By using the equality \(\cos^2(\alpha) + \sin^2(\alpha) = 1\), we can eliminate the parameters \(\eta\) and \(\omega\) in equation (3). Then we get an implicit equation:

\[
\left(\frac{x}{a_x}\right)^{\frac{\varepsilon_1}{2}} + \left(\frac{y}{a_y}\right)^{\frac{\varepsilon_1}{2}} + \left(\frac{z}{a_z}\right)^{\frac{\varepsilon_2}{2}} = 1
\]

Equation (4) can be modified (Solina and Bajcsy) to give the inside-outside function

\[
f(x, y, z) = \left[\left(\frac{|x|}{a_x}\right)^{\frac{\varepsilon_1}{2}} + \left(\frac{|y|}{a_y}\right)^{\frac{\varepsilon_1}{2}} + \left(\frac{|z|}{a_z}\right)^{\frac{\varepsilon_2}{2}}\right]^{\varepsilon_1}
\]

The exponent \(\varepsilon_1\) forces equation (5) to grow quadratically instead of exponentially, ensuring proper convergence during model recovery (Bajcsy and Solina 1987).

The inside-outside function (5) determines where a given point lies relative to the superquadric surface. If

\[
f(x_u, y_u, z_u) = 1, \text{ then } x_u, y_u, z_u \text{ is on the surface,}
\]

\[
f(x_u, y_u, z_u) < 1, \text{ then } x_u, y_u, z_u \text{ lies inside the surface,}
\]

\[
f(x_u, y_u, z_u) > 1, \text{ then } x_u, y_u, z_u \text{ lies outside the surface.}
\]

LV SURFACE EXTRACTION

In our research, we utilize LV data acquired by a high-speed computed tomography (CT) scanner (Robb and Hoffman 1983; Acharya and Heiferman 1987). The data sequence consists of 16 volumetric (128 by 128 by 118) images taken through the heart cycle. The motion of the left ventricle is more important for medical diagnosis, so we concentrate on its inner surface data. This raw data is already thresholded: 1 indicating the heart muscle, and 0 indicating the inside and the outside of the LV (Figure 1). We applied a 3-D edge detector suggested by (Zucker and Hummel 1981) to our 3-D data to extract the inner surface points of the LV (Figure 2). This edge operator is based on the fact that, since different objects give rise to different image intensities, a local operator that responds strongly to intensity differences should be applied to every point in the image.

DATA FITTING

The inside-outside function defines the point position relative to the superquadric surface in an object centered coordinate system \(x_u, y_u, z_u\). The LV surface data points are expressed in a world coordinate system \(x_w, y_w, z_w\). We move the LV data points to the center of the world coordinate system with a homogeneous coordinate transformation matrix \(T^{-1}\):

\[
\begin{bmatrix}
  x_w \\
  y_w \\
  z_w \\
  1
\end{bmatrix} = T^{-1}
\]

The transformation matrix \(T^{-1}\) is the inverse of the transformation matrix \(T\), which first rotates a point, and then translates it from the origin of the world coordinate system (Solina and Bajcsy 1990). By inverting the matrix \(T\) and using Euler angles \((\phi, \theta, \psi)\) to express the elements of the rotational part of the matrix \(T^{-1}\), we get the inside-outside function for superquadrics in general position:

\[
F(x_u, y_u, z_u; a_x, a_y, a_z; a_{11}, \ldots, a_{11}) = 1
\]

The expanded inside-outside function \(F\) has 11 parameters: \(a_x, a_y, a_z\) define the size; \(\varepsilon_1, \varepsilon_2\) define the shape; \(\phi, \theta, \psi\) define the orientation; and \(p_x, p_y, p_z\) define the position in space. The set of all 11 model parameter values will be referred to as \([a_1, \ldots, a_{11}]\).

The superquadric model predicts the position of a point \(x_u, y_u, z_u\) relative to the surface of the model. To model a 3-D LV surface with a superquadric, we can vary the 11 parameters \([a_1, \ldots, a_{11}]\) to get such values that most of the 3-D points will lie on, or close to the model's surface. To find the superquadric model for which the distance from the LV surface points to the superquadric model surface is minimal, we can use the Levenberg-Marquardt method for nonlinear least square minimization (Solina and Bajcsy 1990). Since, for a point on the surface of a superquadric \(F(x_w, y_w, z_w; a_1, \ldots, a_{11}) = 1\), we need to minimize

\[
\sum_{i=1}^{n} (1 - F(x_w, y_w, z_w; a_1, \ldots, a_{11}))^2
\]

To ensure that the iterative fitting procedure will converge to an acceptable solution by starting the minimization in the
correct neighborhood, we can initialize the superquadric parameters \( a_1, a_2, a_3, a_4 \) as follows: we set \( \epsilon_1 = \epsilon_2 = 1 \), such that the superquadric starts as an ellipsoid; we set the position parameters \( p_x, p_y, p_z \) to the center of gravity

\[
(x, y, z) = \frac{1}{N} \sum_{i=1}^{N} (x_i, y_i, z_i)
\]

of all surface data points; we set the extent parameters \( a_x, a_y, a_z \) to the maximum projected length of the principle eigenvector from the centroid (Lagarde and ferris and Whaithe 1989). The principle eigenvector is the eigenvector associated with the largest eigenvalue of the symmetric scatter matrix

\[
\mathbf{S} = \sum_{i=1}^{N} \left[ \begin{array}{c} x_i \\ y_i \\ z_i \end{array} \right] \left[ \begin{array}{c} x_i \\ y_i \\ z_i \end{array} \right]^{T}
\]

We find the initial rotation parameters \( \phi, \theta, \psi \) by aligning the axes of the ellipsoid along the principle moments of inertia of the point about the centroid.

**DISCUSSION AND FUTURE RESEARCH**

By performing the fitting process for each of the images taken at different time instances, we can find the surface point correspondences of the LV during the heart cycle. Then, we can perform the local motion analysis by comparing changes in the Gaussian curvatures as suggested by (Goldgof, Lee, and Huang 1988; Mishra and Goldgof 1990(to appear)). Consequently, we can draw the conclusions concerning the applicability of superquadrics to static LV surface modeling.

To our knowledge, this is the first attempt to apply the superquadrics to modeling of the LV and use it in motion analysis. Even though the superquadrics have proven flexible enough to model a wide class of objects, they do not take into account the motion of the nonrigid object. To perform dynamic motion analysis, we need a time parameter in the superquadric function. Recently, Hyperquadrics (Hasson 1988) has been suggested as a solution for modeling smoothly deformable realistic shapes. Hyperquadrics equation

\[
\sum_{i=1}^{N} \left[ a_{ix}^2 + a_{iy}^2 + a_{iz}^2 \right]^{\epsilon_i} = 1
\]

describes a shape in space one dimension higher than the space we are working in. Hyperquadrics in three dimensions would have four variables \((x, y, z, t)\), and would produce a specific surface for each value of \( t \). Therefore, hyperquadrics can be used to perform dynamic motion analysis by analyzing the changes in model parameters as a function of time. Since hyperquadrics are generalization of the superquadrics, superquadrics can be used as the initial estimate of the hyperquadrics. However, fitting of hyperquadrics is a more difficult problem since we do not have an inside-outside function. Our future research will investigate the application of hyperquadrics to shape and motion modeling of nonrigid objects.

**Acknowledgements**

This research is supported by NSF grants IRI-8817776 and IRI-8900798 (Research Experiences for Undergraduates) and in part by USF/DSR Research and Creative Scholarship grant 2108-931-R0, and FHTIC grant 2108-085-10.

**REFERENCES**


The Heuristic Control of Logic Programs

Timothy Colburn
Department of Computer Science
University of Minnesota, Duluth
Duluth, MN 55812

Jim Carciofni
Honeywell Systems and Research Center
3660 Technology Drive
Minneapolis, MN 55418

Abstract

We describe our version of LogLisp, a language for combining logic programming and heuristic search techniques. By allowing programmer control over the method of deduction tree search, efficient heuristic search of the corresponding abstract state space is possible. This control is made available through a simple and intuitive non-logical language extension which allows general problem solving code to remain intact while experimenting with various heuristics. Judicious use of this extension enables the programmer to tune the logic program interpreter itself without "hacking" the code.

Introduction

In this paper we report on an AI language which is an experiment in combining heuristic problem solving and logic programming. The experiment is not merely to build heuristic search control into logic programs themselves, but to build and experiment with a logic program interpreter which has heuristic control capability built into it. Because of this built-in capability, logic programmers can in many cases clearly separate the tasks of writing clear, intuitive logic programs, and making them execute efficiently. The latter task is one of encoding heuristics which are incidental to the general problem solving code, and only serve to direct the interpreter in its traversal of the deduction tree. How efficiently a deduction tree is searched is a measure of how well reasoning heuristics in the problem domain have been encoded. For a certain type of logic programming problem, then, Kowalski's famous dictum [Kowa79a] can be extended to say Algorithm = Logic + Heuristic.

Thus a logic programming language with built-in tools for optimizing a search space is well suited for problems in which the chief knowledge engineering task is encoding heuristics. Our language, a version of LogLisp, originally developed by Robinson and Sibert [Rob82], allows a programmer to take advantage of efficient depth-first search of the deduction tree underlying a program, while also allowing, at the abstract level, heuristic searching of the corresponding problem state space. The language accomplishes this by providing a programmer-invoked mechanism for suspending computation and resuming on the basis of a programmer-defined heuristic. What distinguishes the language from Robinson and Sibert's original implementation is the amount of user control over deduction tree processing, resulting in the ability to tune the search without "hacking" the code. We believe this to be a natural and positive extension of Robinson and Sibert's original vision.

The paper is divided into four parts. Part 1 quickly reviews logic programming and describes the action of a traditional depth-first logic program interpreter. Part 2 points out the possible limitations of depth-first search through a simple blocks world planning example. Part 3 describes Robinson and Sibert's version of LogLisp, in which deduction tree node suspension is introduced. In Part 4 we describe our LogLisp language extension and give some preliminary test results using the example from Part 2.

1 Depth-First Search in Logic Programs

We here describe logic programming only to the extent necessary to explain the depth-first approach to logic program interpretation. For an introduction to logic programming and its theoretical foundations, see [Hog84] and [Lio84], respectively. For a detailed treatment of the connection between logic, problem solving and computer science, the reader is referred to [Kowa79b].

The value of logic programming lies in its ability to give a declarative interpretation to programs. That is, a logic program can be viewed not as instructions, but as declarations of relations among objects, or facts, and rules for inferring new facts from old ones. To solve a problem in logic programming, a programmer needs to be explicit only about these facts and rules, not the procedural details of how to go about managing the resources of the computer to represent and manipulate these facts and rules. Since the underlying system tends to these details, the programmer is free to concentrate on the important parts of the problem itself.

In logic programming facts and rules are represented using Horn clauses, or restricted forms of predicate calculus formulas. In LogLisp, Horn clauses are represented as lists of predications, where a predication is a list whose first element is a predicate name and whose remaining elements are arguments.
For example, we can represent facts about relative birth dates by

\[
\{(\text{born-before fred otto})\}
\]
\[
\{(\text{born-after butch fred})\}
\]

These clauses represent the facts that Fred was born before Otto and Butch was born after Fred. General rules inferring relative ages from birth dates can be represented by:

\[
\{(\text{older-than } ?x ?y)\}
\]
\[
\{(\text{older-than } ?x ?y)\}
\]
\[
\{(\text{older-than } ?x ?y)\}
\]

In these rules it is understood that \( ?x \) and \( ?y \) represent universally quantified variables, and the first predicate of such a list represents the conclusion part of a rule and the rest represent its antecedents or \emph{goals}. Thus the first rule says that for any \( ?x \) and \( ?y \), if \( ?x \) was born before \( ?y \) then \( ?x \) is older than \( ?y \). Facts are regarded as conclusions without antecedents.

Horn clauses whose conclusions share the same predicate constitute a \emph{logic procedure}. The collection of logic procedures making up a logic program is also called a \emph{knowledge base}. The knowledge base we have given so far contains four clauses which embody three logic procedures. An \emph{input query} is a list of predicates, representing a conjunction of initial predicates to prove from the knowledge base. The object of logic program execution is the substitution of terms for logic variables in an input query which makes it deducible from the knowledge base.

The process of logic program execution is a series of attempts at matching the structure of a current predicate to be proved against the structure of the conclusion predicate of some clause in the knowledge base. The power of logic programming comes from (1) the ability to perform this matching by binding logic variables in a process called unification, and (2) remembering these bindings while proving subsequent goals through an overall inference mechanism called resolution.

Using the knowledge base above, the input query

\[
\{(\text{older-than fred ?z})\}
\]

starts a process which attempts to find substitutions for \( ?z \) which make the query true. The query constitutes an initial goal list which will change as the deduction proceeds. Along the way, the current bindings of variables are remembered by storing them in a growing \emph{environment} of bindings. Figure 1 depicts the process. Note that when a goal matches a clause head, any antecedents of the clause replace the goal in the goal list and any resulting variable bindings are added to the environment. Note also that a solution is found when the goal list becomes empty. The data area for this process can be thought of as a deduction tree of nodes containing goals and logic variable bindings. The branching in the tree structure represents where there are two separate clauses whose heads match the initial goal, corresponding to the two solutions to the query.

One way to implement a logic program interpreter is to follow one path until it leads to a success or failure, backtrack to the point where it had a choice in matching clauses, and continue. Such an interpreter does a depth-first search of the deduction tree, since the first descendant of a node is completely searched before exploring its next sibling.

In order to clearly describe LogLisp’s approach to logic program interpretation, we first present an algorithm for depth-

![Figure 1: A backtracking approach to logic program interpretation](image)

The algorithm for a depth-first logic program interpreter is given in Figure 2. This algorithm embodies a classic Hor

![Figure 2: Algorithm for a depth-first logic program interpreter](image)
clause interpreter. Note that the depth-first nature of its search is inherent in the algorithm's use of recursion; since it recurses in order on each unexplored clause for a node, the left-most child of a node is completely explored before moving to its sibling.

Of course, the first usable and efficient language incorporating this style was Prolog. By adding a number of built-in predicates to facilitate arithmetic and I/O, Prolog achieved for predicate logic the status of a general purpose programming language. From an implementor's standpoint, efficiency is gained by managing the deduction tree and variable binding data structures as stacks, allowing backtracking to be accomplished through stack pops. Furthermore, the addition of the cut operator allowed Prolog programmers to prune the tree by taking advantage of the language's efficient backtracking methods.

2 Limitations of Depth-First Search

But no matter how a logic programming language is embellished, if it implements the deduction algorithm above it is constrained to doing depth-first search of the deduction tree with backtracking. Thus, depending on certain incidental characteristics of the knowledge base, like the order in which predicates are stated, a query may require that a large portion of the deduction tree be unsuccessfully searched before a solution is found.

As an example, consider a logic program adapted from one given by Sterling [Ster84] for planning in the blocks world. The problem is to come up with a list of actions which will transform an initial state of the blocks world into a final state. For example, the shortest list of actions to transform the initial state into the final state of Figure 3 is

\[
\begin{align*}
\text{(move a from b to r)} & \quad \text{(move b from p to q)} \\
& \quad \text{(move a from r to b)}.
\end{align*}
\]

![Figure 3: Initial and final states in the blocks world](image)

Sterling's solution recognizes logical levels of problem solving. The meta-level is concerned only with general plan formation, regardless of the problem domain, and does not even know about the blocks world. This level in effect simulates performance of actions in the blocks world. We can write a meta-level planner in our Horn clause notation as follows:

\[
\begin{align*}
\text{(make-plan ?plan)} \\
& \quad \text{(initial-state ?start)} \\
& \quad \text{(final-state ?final)} \\
& \quad \text{(state-trans ?start ?final nil nil ?plan)} \\
& \quad \text{(state-trans ?cur ?final ?states ?acts ?acts)} \\
& \quad \text{(equal ?cur ?final)} \\
& \quad \text{(state-trans ?cur ?final ?states ?acts ?dum)} \\
& \quad \text{(update ?cur ?facts ?states ?next)} \\
& \quad \text{(state-trans ?next ?final (?cur . ?states) (?act . ?facts) ?dum)}
\end{align*}
\]

These clauses describe the action of a general planner. state-trans is a recursive procedure which begins with the initial state and continually generates a new current state until it is equal to the final state. It does this by calling update, which does the actual work of generating the new states and associated actions, being careful to avoid circularity. These clauses could be used for generating plans in any computational world; it is the definition of action and transform which interfaces the meta-level to the object-level, which in this case happens to be concerned with methods and facts involving blocks and places.

In this particular domain, a state is a list of predications indicating which blocks are on top of others and which are clear. In our example, the initial and final states are represented by the clauses:

\[
\begin{align*}
\text{(initial-state ((on a b) (on b p) (clear a))} \\
& \quad \text{(clear q) (clear r))} \\
\text{(final-state ((on a b) (on b q) (clear a))} \\
& \quad \text{(clear p) (clear r))}
\end{align*}
\]

An action in the blocks world is to move a block from one block or place to a different block or place, provided the destination is clear. The current state is needed to provide context for determining whether the move is legal. An action can be described by the following clause:

\[
\begin{align*}
\text{(action (move ?x from ?y to ?z) ?state)} \\
& \quad \text{(on ?x ?y ?state)} \\
& \quad \text{(clear ?x ?state)} \\
& \quad \text{(clear ?z ?state)} \\
& \quad \text{(diff ?x ?z)}
\end{align*}
\]

where the required goal predicates are defined as:

\[
\begin{align*}
\text{(on ?x ?y ?state)} & \quad \text{(member (on ?x ?y) ?state)} \\
\text{(clear ?x ?state)} & \quad \text{(member (clear ?x) ?state)} \\
\text{(diff ?x ?y)} & \quad \text{(not (equal ?x ?y))}
\end{align*}
\]

Procedure transform creates the state resulting from a legal move of ?x from ?y to ?z by deleting the predicates (on ?x ?y) and (clear ?y) from the current state and adding the predicates (on ?x ?z) and (clear ?y) to the current state.

This logic program, together with the necessary utility predicates, can be used to generate the shortest solution. Figure 4 shows the search space for our simple problem. Note that a depth-first, left-to-right search of this tree, implemented by our Prolog-like logic program, must first find 39 longer solutions before finding the shortest, which happens to be the last. This ordering of solutions depends upon the ordering of predicates within the current state, which in turn depends upon the ordering of predicates within the initial state. Thus program performance might be improved by hacking the coding of the initial state, but such recourse is antithetical to the goals of logic programming.

3 Robinson and Sibert's Version of Log-Lisp
The original designers and implementers of LogLisp, though impressed with the success and widespread use of Prolog, nevertheless desired an implementation of logic programming that was not constrained to one particular control regime:

...[T]he procedural interpretation of Horn clauses within a LUSH-resolution theorem prover permits, but by no means requires, a backtracking process for the exploration of alternative computations. These computations can (as in LogLisp) be developed in quasi-parallel, thereby avoiding the impression that logic programming is (like PLANNER) about advancing and retreating, trying, failing and trying again, and now and then succeeding. [Robi82]

Rather than executing a logic program by following a deduction tree branch until it either succeeds or fails, Robinson and Sibert treated a deduction cycle as the computation of all of a node's immediate descendants. These descendant nodes were then added to a "waiting heap" of pending nodes, from which one was selected to begin the next deduction cycle. This selection was based on the lowest "solution cost", an estimate involving a linear combination of the number of goals left to prove in a node and its depth in the deduction tree. An algorithm for Robinson and Sibert's LogLisp interpreter thus requires the following additional functions:

- `add-to-heap` computes the solution cost of a node and inserts the node into a waiting heap based on this cost. It is called for side effect only.
- `remove-from-heap` returns the node with lowest solution cost on the heap, removing it from the heap as a side effect.

The algorithm is given in Figure 5.
We began implementing this algorithm with a performance goal of 500 LIPS (logical inferences per second) on a VAX 11/780. We quickly found that the overhead of adding to the heap for every successful goal unification made this goal unattainable, with our best efforts resulting in run times of barely 100 LIPS. We coveted the run-time technologies of Prolog, but since they depended heavily on depth-first traversal of the deduction tree, they were out of reach so long as heuristic suspension of nodes took place at the time of every successful unification. It was then that we discovered a compromise between unrestricted backtracking and unrestricted heuristic node suspension which we believe results in a logic programming system with the correct amount of opportunity for heuristic control.

4 Our Version of LogLisp

It occurred to us that, within the logic programming paradigm, the correct level of abstraction at which to apply heuristic node suspension was not at the level of the run-time deduction tree, but rather at the level of the problem search space. The difference between levels is considerable. A problem search space, like that given in Figure 4, has whole states for nodes, while a deduction tree has clause goals and environments for nodes. The deduction tree corresponding to Figure 4 is much larger than the state space, and furthermore is at a level of detail of no use to the typical programmer. If a programmer wants to implement a solution cost heuristic, he or she needs to do so with respect to problem states, and not deduction tree nodes. In order to offer this in a logic programming language, we have to compromise its declarative nature, but only slightly: we have to make deduction tree node suspension an option available under user program control. Furthermore, we need to allow the programmer control over how the solution cost heuristic is calculated.

For example, think of suspending each state in our blocks example on an abstract state heap, not a heap of deduction tree nodes, and selecting from this heap the state whose distance from the initial state is shortest. Since this distance is equal to the number of block moves made, the first solution found is guaranteed to be the shortest. In the context of our from clause program, the logic procedure state-trans keeps track of the list of moves made so far in the variable ?acts. The length of this list is therefore an appropriate criterion by which to order the states on the state heap. By allowing the user to interrupt the depth-first traversal of the deduction tree just before the next state transition is attempted (and only then), and to suspend the current deduction tree node according to this criterion, we can simulate a heuristic search of the abstract state space while preserving speedy depth-first processing of the vast majority of the actual deduction tree space.

In our version of LogLisp we allow this through the use of the suspend form. This form takes one argument, an integer solution cost, and adds the current deduction tree node to the waiting heap on the basis of this cost. Processing then resumes with the node on the heap with the smallest solution cost. We can convert our blocks program from a depth-first searcher to a heuristic searcher by judiciously placing one call to suspend in one logic procedure:

\[
\begin{align*}
&((\text{state-trans} \ ?\text{cur} \ ?\text{final} \ ?\text{states} \ ?\text{acts} \ ?\text{facts}) \\
&\quad (\text{equal} \ ?\text{cur} \ ?\text{final})) \\
&((\text{state-trans} \ ?\text{cur} \ ?\text{final} \ ?\text{states} \ ?\text{acts} \ ?\text{fun})) \\
&\quad (\text{update} \ ?\text{cur} \ ?\text{fact} \ ?\text{states} \ ?\text{next}) \\
&\quad (\text{suspend} \ ?\text{!length} \ ?\text{acts}) \\
&((\text{state-trans} \ ?\text{next} \ ?\text{final} \ ?\text{cur} \ ?\text{states}) \\
&\quad (?\text{fact} \ ?\text{acts} \ ?\text{fun}))
\end{align*}
\]

The \texttt{!} syntax tells LogLisp to interpret the next form as a Lisp form with logic variables. Here it allows the LogLisp function \texttt{length} to be called using the list bound to \texttt{?acts} as its argument.

The introduction of this suspend form does not alter the declarative semantics of the rest of the program, nor does it prune the deduction tree in any way. But it does allow user control of the deduction tree search. The program will find the best answer with or without this form, but with it the program will find the answer 55 times faster (see Table 2). In a sense the form serves to tune the interpreter without hacking the code.

The algorithm embodied by our implementation of LogLisp is given in Figure 6. Note that the outer loop picks nodes off

\begin{verbatim}
procedure deduce:
begin
while not null(heap) do
begin
node := remove-from-heap;
if null(node.glist) then
report-solution(node)
else
explore(node)
end
end
\end{verbatim}

recursive procedure explore(node);
begin
if null(node.glist) then
report-solution(node)
else
while has-unexplored-clauses(node) do
begin
goal := head(node.glist);
if suspension-requested(goal) then
add-to-heap(node)
else
begin
bindings := unify(goal, head(next-clause(node)));
if env \not\in fail then
explore(make-new-node(node, bindings))
end
end
\end{verbatim}

Figure 6: Algorithm for a user-directed heuristic logic program interpreter
lead to suspension, at which point the heap is consulted for where to process next. Thus infrequent but judicious calls to suspend result in processing which is nearly depth-first, but informed enough by domain knowledge to significantly optimize the search. In our example, the placement of one suspend form in effect enabled a breadth-first search of the abstract state space tree, while allowing practically all of the concrete deduction tree to be processed depth-first. Without the form, both the state tree and the deduction tree are processed entirely depth-first, with much wasted processing the result.

The amount of blocks world domain knowledge inherent in calculating the number of moves made so far is, obviously, not much. But since our implementation of LogLisp allows the user to calculate a solution cost, more detailed knowledge can be incorporated to inform the heuristic. For example, a place or block whose final state is clear and whose current state is not clear will cost more than if its current state is already clear. Table 1 shows how one might assign costs for the various combinations. By writing a straightforward cost function in Lisp to total these costs for all places and blocks in the current state, we can effect a true best-first search of the state space by replacing the suspension in state-trns with

\[
\text{(suspend #f(cost ?cur))}
\]

Again, since none of the problem-solving code is altered, the use of node suspension here serves only to optimize the interpreter, but with impressive results, summarized in Table 2.

<table>
<thead>
<tr>
<th>Search mode</th>
<th>User CPU time in milliseconds</th>
<th>Speed-up factor over depth-first</th>
</tr>
</thead>
<tbody>
<tr>
<td>depth-first</td>
<td>1600580</td>
<td></td>
</tr>
<tr>
<td>breadth-first</td>
<td>2920</td>
<td>55</td>
</tr>
<tr>
<td>best-first</td>
<td>960</td>
<td>157</td>
</tr>
</tbody>
</table>

Table 2: Comparative times and speed-ups for the blocks example.

Conclusion

Our version of LogLisp is not intended to compete with Prolog. Instead, we view it as a logic programming platform on which to code knowledge-based applications that require a good deal of abstract state space searching as part of their reasoning mechanisms. Using LogLisp, such a space can be searched heuristically while large subtrees of the corresponding concrete deduction tree are searched depth-first. This allows applications to exploit Prolog’s approach to memory management yet retain the flexible control of heuristic reasoning in a way which satisfies one’s taste, if one has it, for declarative clarity.

References


SEARCHING WITH CONSTRAINTS: A SCHEDULING APPLICATION

Randy Beth Pollack
Department of Computer Science
Florida Institute of Technology
Melbourne, Florida 32901

Eugene A. Pierce
NCR Corporation World Headquarters
1700 South Patterson Blvd.
Dayton, Ohio 45479

ABSTRACT

The problem domain of faculty scheduling is used to investigate the utilization of constraints and constraint processing as a knowledge representation facility in the context of search. A search strategy is used in which constraints enable the problem solver to focus on the most promising segment of the search space.

The network improves search efficiency by partitioning the search space into smaller attribute specific subsets which can be searched more easily than the entire unconstrained search space. It provides additional benefits in that it enables data objects to maintain knowledge of other data objects; if an object in the network is modified it can propagate the modification to other objects with which it has relationships. The scheduling procedure results in a network that may be utilized for schedule repairs without starting the process over from the beginning.

INTRODUCTION

Many artificial intelligence applications using search are limited in size and power by the combinatorial explosion that occurs in the processing of a large search space. The objective of this research was to investigate the utilization of constraints and constraint processing as a knowledge representation framework to facilitate problem solving in the context of search.

Examples include work by Fox et al. (1982), Smith (1987) and Stefik (1981). Much of the work relating to scheduling has been in the area of job shop scheduling or other issues related to production and resource allocation. Little has been done in the area of personnel scheduling, otherwise known as timetabling.

One of the few expert systems in this area is described by Murphy (1987) in the realm of faculty and class scheduling, the same domain used in the research reported in this paper. Murphy’s Timelog program is essentially an intelligent spreadsheet where the normal spreadsheet functions interact with a set of Prolog rules containing the system’s constraints. It acts as an aid to a human scheduler rather than as an independent scheduling system. Even with a limited number of teachers (8), the program’s main problem is high memory usage and slow performance.

The approach that we have taken focuses more on the search aspect of the problem. Constraints are used as heuristics which enable the problem solver to focus on the most promising segments of the search space, resulting in more efficient generation of solutions. A depth-first search strategy is followed which uses the constraints to choose among the paths which could be followed. Backtracking is used when no further progress can be made down a particular path.

The constraint network improves search efficiency by partitioning the search space into smaller subsets which can be searched more easily than the entire unconstrained search space. The network provides additional benefits in that it enables data objects to maintain knowledge of other data objects; if an object in the network is modified it can propagate the modification to other objects with which it has relationships.

Proceedings of the 3rd Florida Artificial Intelligence Research Symposium, 1989, by the Florida AI Research Society

c. 1989 FLAIRS 0-9620-1732-9/90/0360-0019

19
THE APPLICATION DOMAIN

In order to demonstrate the feasibility of this approach, a search program was written to perform the task of assigning courses to a college department’s faculty. Unlike the Timelog program described above, this is an independent scheduler that does not require human interaction. The system was written using the SCOOPS (object oriented) extension of the language Scheme.

The knowledge used by the system includes each quarter’s course offerings (course and section numbers and times offered), and the names, teaching loads and preferences of the instructors. Each faculty member may express preferences by ranking all courses on a 5-point scale where the extremes represent most preferred (1) and least preferred (5) courses, and the middle ranking (3) represents no preference. Similar rankings may be input for the time slots in which courses are offered. The scheduler has additional information on the number of different course preparations each instructor should be assigned; for example, it may be told that a particular instructor should be given three courses during a particular quarter, but no more than two different preparations. From this information, a constraint network is built.

Each instructor is represented as an object in the network, as is each course section. Based on the preferences he or she has entered, each instructor is associated with lists of courses that meet those constraints to varying degrees. Omitted from the lists are those sections which have been assigned a ranking of 5 (least preferred) based on either time or course number.

These course lists are, in effect, attribute specific search spaces which are subsets of the complete search space (the entire list of course offerings). However, there is clearly going to be a large degree of overlap among these smaller search spaces, and the constraint network functions to maintain consistency among them. If a course is assigned to an instructor, that assignment affects the search space of another instructor who has that course included in his or her list of most preferred courses. While the course remains in both search spaces, it obviously cannot be assigned to more than one individual and the constraint network performs the task of monitoring these assignments.

The scheduling phase of the program attempts to assign each instructor a schedule consisting of those courses which are most preferred rather than considering courses randomly for inclusion into the instructor’s schedule. In other words, a filtering process has already taken place which eliminates consideration of irrelevant data during the search. The filtering process also ranks the courses remaining in the search space according to instructor preferences, so courses that most closely meet the constraints will be considered first.

When attempting to formulate a schedule for an instructor, the system first chooses as many sections as possible from that individual’s list of most preferred courses. If all courses on the ‘most preferred’ list have already been assigned, backtracking takes place. There is an attempt to reschedule other instructors who have already been assigned those courses, replacing them with others that are equally preferred. The backtracking algorithm will not take a course away from a previously scheduled instructor if an equally preferred course cannot be substituted. This feature of the algorithm allows an implicit ranking of instructors; those scheduled first will be more likely to get their most preferred assignments.

The scheduling algorithm also considers the number of preparations assigned to each instructor in its initial assignments and backtracking. If an instructor has three sections but only two preparations, a course with multiple sections must be used to satisfy those constraints. Therefore, the scheduling algorithm will give preference to that instructor over another instructor who has no need for multi-section courses when making assignments involving such a course.

If a ‘most preferred’ course cannot be found for an instructor by the backtracking algorithm, the scheduler will assign a less preferred course. Depending on the number and type of courses offered and the particular time and course constraints imposed by an instructor, it may happen that there is no set of courses available to fill out an instructor’s schedule and meet his or her constraints. In this case, the schedule for that instructor will be incomplete. The scheduler will not assign courses that violate the stated constraints associated with an instructor.

The constraints act as heuristics, guiding the search along paths that should be the most productive. However, this approach differs from traditional heuristic search in that it isolates the
application of heuristic knowledge away from the search algorithm. Rather than using an evaluation function which must be applied as the search executes, the heuristics are applied before the search begins to associate each instructor with those possible course assignments that best meet his or her constraints. Therefore, the search/scheduling portion of the program has a much better runtime performance than it would have if each potential course assignment had to be evaluated as the search progressed.

The final output of the scheduler, shown in Figure 1, is a network in which instructor objects are linked to the course sections to which they have been assigned. This graph is a secondary data structure which can be used after the initial scheduling to accommodate changes that may be made to the data. For example, if a course is dropped from the quarter’s offerings or an instructor is dissatisfied with the schedule, changes can be made that will propagate easily through the network. It is not necessary to reconstruct the schedule completely in response to each change; utilities can be written to allow repairs to an existing schedule.

RESULTS

Preliminary testing was done with 20 instructors and four different sets of instructor constraints and course offerings that varied the amount of stress placed on the scheduler. Stress is induced by reducing the fit between course offerings and instructor constraints, placing high demand on particular course offerings. This results in more backtracking than a situation in which there is a large selection of courses that fit instructor preferences, or in which instructor constraints are minimal.

In each case, a schedule was generated successfully. The situation with the most stress completed in approximately 10 minutes, while the least stressful configuration took only 6 seconds.

In the more difficult configurations, one or more instructors were left with incomplete schedules due to the fact that the problem was overconstrained. There existed no solution in which all instructors could be completely scheduled, given the course offerings and instructor constraints. However, it should be noted that the more stressful configurations were judged to be unrealistic in terms of an actual department’s offerings and instructor constraints, but were used simply to test the performance of the system.

FUTURE WORK

The ability to repair schedules, referred to above, has not yet been incorporated into this system. While the network produced by the program
clearly lends itself to this type of manipulation, the utilities have yet to be written which will implement this feature.

A provision in which constraints may be relaxed is an addition to the system that would solve the problem of an overconstrained situation as described above. This would involve the incorporation of an algorithm to evaluate the constraints and then choose the one(s) of least practical importance. These already existing scale of instructor preferences could be used in such an implementation to help in this decision, but certain constraints must necessarily be marked as inflexible. An instructor must not be scheduled into two classes that meet at the same time.

Another feature that would improve the resulting schedule would be the addition of 'critics'. A set of rules could be used to evaluate each instructor's schedule in terms of possible problems that were not addressed by that individual's list of preferences. For example, scheduling of an early morning and late evening course on the same day might be noted by a critic which would force rescheduling of one of those sections. The critics could operate either during the scheduling process or after a first draft schedule has been constructed.

Further work in this and other domains should be attempted to determine if this model can be successfully extended to larger problems. If so, the development of a shell to provide a general framework for solving scheduling and time-table problems should be considered. It remains to be shown whether a generic approach can adequately represent a majority of scheduling applications or if it will result in too great a loss of flexibility and/or efficiency.

SUMMARY AND CONCLUSIONS

In summary, the application domain of faculty scheduling demonstrates that constraint networks can be used to assist in problem solving in the context of search. The key effect of this integration is that the search space can be intelligently segregated so that search can execute more efficiently. In addition, the data structure produced by the search/constraint satisfaction combination is amenable to later modification.

REFERENCES


A PARALLEL KNOWLEDGE-BASE SEARCH STRATEGY

Patrick O. Bobbie
Division of Computer Science
University of West Florida
Pensacola, FL 32514

ABSTRACT

Knowledge-base (KB) systems possess natural concurrent parts that may well translate into parallel environments. At the same time, portions of a KB system that lack inherent concurrency, regardless of the parallel machines available, can not take advantage of more than one machine. Thus, the central goal of distributed KB problem-solving is the degree to which computational efficiency may be achieved through parallelism. The degree to which the efficiency may be achieved is predicated on the format for representing and implementing the KB system. PROLOG is a declarative language, well suited to describing KB systems whose evaluation is independent of procedural enactment. PROLOG programs are composed of three constructors: rules, facts, and queries. Since the execution of PROLOG programs also suggests parallel evaluation and requires referential transparency of variables, it is suited for parallel environments. Thus, the mix of PROLOG-based Artificial Intelligence (AI) systems and parallel computing is not coincidental. The focus of this paper is on strategies for searching rules and facts in PROLOG-based KB systems. We discuss methods for modeling, partitioning, and clustering KB data. The intent is to reorganize the KB into clusters of logically related rules, facts, and queries that would map onto parallel environments to reduce search time.

I. INTRODUCTION

The rationale for investigating parallel or distributed KB problem-solving includes the desire to accelerate searching, reasoning, and inference; increase availability of knowledge data through redundancy and replication of parts of the KB; modularity; and autonomous coordination of several concurrent AI tasks. To achieve these goals, two basic approaches have emerged, namely Distributed AI (DAI) and Parallel AI (PAI).

DAI facilitates the integration of diverse domain knowledge through semantic mappings of the domains (Bodn and Gasser 1988; Findler and Lo 1986). DAI also supports the coordination of autonomous problem-solving agents in AI. For example, consider the use of a knowledge representation scheme for specifying state-oriented behavior (dynamic) and static attributes of a real-time concurrent software system. To include a specification of modular concurrency of the system, the specifications might include abstract descriptions of the logical configuration and time parameters of the target distributed architecture. It might also be desirable to include rules for mapping the logical functions of the system onto the logically specified hardware configuration and time parameters. The mapping procedure should be carefully done without constraining the portability of the resultant real-time software across other machines. Thus, in this example, rule-based specifications might be constructed from (1) the software domain and (2) the hardware domain. The critical task in developing such DAI specifications is defining the precise meta-rules for unifying both hardware and software requirements specifications for the real-time software. Furthermore, DAI language mechanisms (e.g. rule-based semaphores) are to be incorporated in the specifications to indicate the coordination of communication between the specified software and hardware modules.

On the other hand, AI computational efficiency is a function of parallelism inherent in a given problem. The task of finding parallelism in a domain knowledge is fundamental to PAI research. As a consequence, the development of methodologies for determining parallelism in AI systems has lately become a significant research issue. In addition to efficiency, PAI research is targeted toward systems where parallelism is exploited to reduce complexity (Fennel and Lesser 1977). The focus of this paper is on a PAI approach to exploiting parallelism in knowledge-bases for an efficient search and reasoning.

In section II, we present an overview of our methods. Section III is on KB representation and analysis methods. Furthermore, we discuss our methods of modeling and decomposition; and show through an example, how resulting KB clusters are constructed to facilitate a parallel search.

II. METHODOLOGY

To improve the efficiency and performance of AI computations, finding parallelism in domain knowledge is one of several approaches. Other techniques toward the attainment of efficiency is to take advantage of parallelism in hardware, employ run-time support of underlying (distributed) operating systems, or reorganize KB efficiently by analyzing language constructs used for knowledge representation. Our approach follows the latter case. Languages like Concurrent Prolog (Shapiro 1983) and Actor-based systems (Agha 1986) provide a linguistic format for representing and expressing parallelism in KB. Although a language like standard sequential PROLOG does not provide constructs for expressing parallelism, however, it has an easy-to-understand expressive format for KB representation. PROLOG is a rule-based language with three constructors: rules, facts, and queries. To detect parallelism,
the syntactic structures of the rules, facts, and queries are analyzed and preprocessed. Subsequently, a mechanism is devised to control a parallel search of the processed program. Several of the mechanisms in use involve insertion of language constructs in the program to convey semantics of parallel evaluation. Currently, there are several types of language-based parallelism. Among these are AND-parallelism, OR-parallelism, and argument-parallelism (Conery 1987).

We focus on standard PROLOG representation of KB. Specifically, we exploit a different kind of parallelism, namely parallelism due to clustering of rules, facts, and queries. To this end, a rule-based KB is clustered into subsets of rules, facts, and queries (KB objects). The clustering process involves a syntactic matching of the rules, facts, and queries for the occurrences of common variable and predicate names. Thus, the objects are analyzed for predicate dependencies or relationships using pattern-matching methods. This determines the presence of dependencies and the absence thereof. The dependency relation used in modeling the rules is the predicate-predicate type. The matrices are partitioned into submatrices where each submatrix is equivalent to a cluster of KB objects.

The rationale for using matrix models as a knowledge analysis tool is twofold. The first reason is a consequence of a successful, earlier research on using matrix model partitioning schemes to group entities based on common relational attributes (Bobbie and Urban 1989). The second reason is that, in general, matrices are relational structures which serve as useful tools for information processing. Other kinds of matrix-based tools have been used for knowledge analysis (Bobbie et al. 1989; Braun 1989). The goal of the research result reported in (Braun 1989) was to structure and organize KB. Our goal goes beyond KB restructuring by generating additional meta-rules for 1) controlling rules/facts search in parallel environments and 2) seeking opportunities to minimize interprocessor communication overhead.

III. KB REPRESENTATION

The human mind is endowed with a capacity and marvel of efficiency in handling large sets of relationships and objects. Every single concept conveys a multitude of relationships, yet the mind is capable of assimilating variations of associated semantics without attention. The mind, therefore, makes humans capable of handling many natural and computer languages and possess enormous searching and reasoning power. However, computers lack this capability and, therefore, require explicitly defined relationships and objects of a given domain before searching and reasoning can be accomplished. Knowledge is elicited from a domain expert and stored in KBs for further processing (Bobbie et al. 1989).

Knowledge-bases can be expressed as rules, facts, and queries; using the frame model; object-oriented model; semantic model; or entity-relationship model. We present two of these schemes to motivate the discussion. Figures 1a and 1b are listings of rule-based and frame-based models for KB representation. Both models convey the semantics of logic and, therefore, suitable for AI problem-solving. In the following, we discuss where the two models differ, the advantages and disadvantages, and features that facilitate parallel AI computations.

old_employee(X) :- salary(X,Y), greater(Y, 50K).
fired(Z) :- violate_policy(Z).
retirement(Y) :- age(Y, X), old_employee(Y), greater(X, 60).
pension_support(X) :- retirement(X), old_employee(X), not(fired(X)).
old_employee(Tom),
salary(Tom, 30K),
old_employee(John),
salary(John, 70K),
age(Tom, 55),
age(John, 65),
violate_policy(Peter),
pension_support(Tom),
pension_support(John).

Figure 1a. Rule-based KB Representation

Differences In Representation

Unlike frame models, rule-based models lack a natural clustering structure. However, a structure can be brought to bear on a rule-based system by grouping the rules, facts, and queries. In Figure 1a, the facts and queries may be artificially grouped into two clusters based on the argument-values Tom and John. The rules do not offer immediate syntactic or semantic features for an artificial grouping or clustering. Thus, to search the knowledge-base of Figure 1a, it might be advantageous to have two replicas of the KB rules. One replica is then combined with the "Tom" cluster and the other replica with the "John" cluster. Although a redundancy is thus introduced, the computational efficiency may well be improved. In this simple example, there are no conflicts in the facts of the two clusters. Hence, if the two sets of rules and clusters are assigned to two parallel computing nodes, the search time could be further reduced. Because, each processor's search space is then less than the case when a single processor is loaded with the entire KB.

In Figure 1b, the frame-based model has frame constructors and semantics for representing knowledge base in clusters or classes. Each cluster denotes a topic or entity with associated attributes, constraints, and procedures for implementing violations of constraints. The natural clusters of frame-based systems offer opportunities for concurrent AI computations. However, the degree of parallelism is dependent on the semantics of the problem and how efficiently the model is constructed.

Analysis And Modeling

Unfortunately, there are no realistic opportunities for an artificial clustering of objects in a large rule-based KB system. Besides, the clustering of facts and queries is generally considered as an exploitation of OR-parallelism (Conery 1987). In our approach, the clustering of facts and queries is a natural consequence of clustering of the rules, since the rules use or call the facts (with a procedural semantics). The rules in Figure 1a are clustered based on how the rules are resolved using the dependency on the goals.
FRAME EMPLOYEE
slot1: name
slot2: age
slot3: salary
slot4: violation
constraint: no violation allowed
method: fire employee
END.

FRAME EMPLOYEE INSTANCE
name: Peter
age: 25
salary: 40K
violation: yes
END INSTANCE

FRAME OLD_EMPLOYEE IS-A EMPLOYEE
slot1: name
slot2: age
slot3: salary
constraint: salary > 50K
method: no pension_support
END.

FRAME OLD_EMPLOYEE INSTANCE
name: John
age: 65
salary: 70K
violation: no
END INSTANCE

FRAME RETIREMENT IS-A EMPLOYEE
slot1: name
slot2: age
slot3: salary
constraint: age > 60
method: no compensation
violation: no
END.

FRAME RETIREMENT INSTANCE
name: Tom
age: 55
salary: 30K
violation: no
END

Figure 1b. Frame-based KB Representation

In the definition (or body) and other rules. Furthermore, argument-parallelism is exploited within the resulting clusters. Methods for controlling the execution of goals involving argument-parallelism are discussed in (Chowkwanyun and Hwang 1989). In the sequel, we outline a procedure for analyzing and clustering rules.

First, a list of predicate-names (head of the rules) is compiled. Each predicate-name occurs once in this list. The names of the goals (goal-names) to the right-side of the rules are added to this list. A matrix is then constructed such that the rows are labeled by only the predicate-names. The columns are labeled by the predicate- and goal-names. Furthermore, we employ a pattern-matching algorithm for finding instances where a predicate-name of a rule occurs as a goal-name in other rules in the KB. When such an occurrence exists, a 1 is entered in the corresponding cell of the matrix. Thus, the cell (predicate-name, other-predicate-name) or (predicate-name, goal-name) is assigned the value 1. At the end of the pattern-matching process, other entries are assigned 0 where no matches are found. Hence, the matrices are called binary type. Figure 2a is a relational binary matrix of the rules in Figure 1a. The matrix is partitioned into submatrices where each submatrix is equivalent to a cluster of KB objects, as shown in Figure 2b.

Legend of predicate- and goal-names:
a: age, f: fired, g: greater, o: old_employee,
p: pension_sup, r: retirement, s: salary, v: viol

<table>
<thead>
<tr>
<th></th>
<th>o</th>
<th>f</th>
<th>r</th>
<th>p</th>
<th>s</th>
<th>g</th>
<th>a</th>
<th>v</th>
<th>sums</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>1+</td>
<td></td>
<td>1+</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>f</td>
<td></td>
<td></td>
<td></td>
<td>1+</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>r</td>
<td>1+</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1+</td>
<td>1</td>
<td></td>
<td>1+</td>
</tr>
<tr>
<td>p</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
</tr>
</tbody>
</table>

s 2* 1 1 0 1 2* 1 1

* entries reduced by 1   1+ marked entries

Figure 2a A Model of the Rules in Figure 1a

Legend of predicate- and goal-names:
a: age, f: fired, g: greater, o: old_employee,
p: pension_sup, r: retirement, s: salary, v: violation

<table>
<thead>
<tr>
<th></th>
<th>v</th>
<th>s</th>
<th>o</th>
<th>f</th>
<th>g</th>
<th>a</th>
<th>r</th>
<th>p</th>
<th>sums</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>o</td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>r</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>p</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td>-</td>
</tr>
</tbody>
</table>

s 0 0 0 0 0 0 0

Figure 2b: Partitions of Rules in Figure 1a.

Partitioning

The steps of the matrix partitioning procedure are discussed in much detail in (Bobbie and Urban 1989). Due to space limitation and for completeness of the discussion, we describe the algorithm to motivate its applicability to reorganizing KB data for parallel environments. The procedure begins by summing the non-zero entries in each row and column in the matrix. A new matrix is then constructed, beginning with the first step. In the first step,
the rows and columns which have only one non-zero entries are selected and moved to the lower or upper corners of the new matrix. Once a row and column are selected, the corresponding cell-entry is marked and then moved to the main diagonal of the new matrix. The corresponding row and column indices are deleted and not considered furthermore. The sums of the remaining rows and columns are reduced by one if any of their non-zero entries are deleted as well. When a sum goes to zero, the affected row or column is moved to the bottom-most row or right-most column of the new matrix, respectively. The step is recursively applied to the matrix until there are no rows or columns with sums equal to one. This step establishes a boundary condition for maximal decomposition, as in Figures 2a and 2b.

The second step is a continuation of the first one using the remaining rows and columns in the matrix as input. The aim of this step is to maximize the number of non-zero entries on the new matrix's main diagonal in order to establish the condition for maximal partition. Each of the remaining non-zero entries is considered and selected if the aggregate sum of the corresponding row and column sums is the minima (ties are broken arbitrary). Once selected and marked, the corresponding row and column are moved to the new matrix and the old matrix is adjusted as done in the first step. When the second step is completed, the non-zero entries which are not marked are moved to match their corresponding rows and columns in the new matrix. The theoretical basis of the first and second steps is founded in bipartite graph theory (Harary 1969) and a detailed example is given in (Bobbie and Urban 1989). Figure 2b does not illustrate every aspect of the second step since the model is not sufficiently, large.

The last step involves moving non-zero entries outside the main diagonal of the new matrix to cluster around the main diagonal. We construct a transitive closure of the entries such that there is a transition from an entry, through a main diagonal entry, to one or more other entries. As the entries are moved into the transitive closure sets, the corresponding row and column indices are rearranged in the matrix. Consequently, the construction produces sets of rows and columns that constitute the partitions. An optimal partitioning or clustering indicates having all clusters around the main diagonal, as shown in Figure 2b.

Figure 2b shows two different groups of partitions. A group (G1) of three main diagonal partitions and two adjacent non-main diagonal partitions. A group (G2) of two main diagonal partitions, one of which is bounded by arrows and two adjacent non-main diagonal partitions. Two alternative clusters are constructed. The first alternative is from G1 as illustrated in Figure 3a and the second alternative is from G2, also illustrated in Figure 3b.

**Clustering And Parallel Search**

The clusters are formed by applying heuristic methods to achieve a minimum overlaps of the partitions. The choice of a particular heuristic method, which entails the exchange or shifting of members of the partitions, is tailored toward a given application. The major concern, in this example, is the overhead of interprocessor communication during the search for a rule or fact. Thus, we consider the effectiveness of migrating facts or rules versus the migration of search-command under two alternatives. The rules in the three diagonal partitions (G1) of Figure 2b form the respective clusters C1, C2 and C3 of Figure 3a. The rules in the two non-main diagonal partitions are placed in the respective clusters, C2 and C3, based on their adjacency to the two lower partitions on the main diagonal of Figure 2b.

Similarly, the rules in the two main diagonal partitions (G2) of Figure 2b form the respective clusters D1 and D2 of Figure 3b. Again, the rules in the two non-main diagonal partitions are placed in the respective clusters based on the adjacency.

The following demonstrates that the alternative cluster are equivalent. The meta-rules for controlling the search space and exploiting clustering alternatives for reducing interprocessor communication are given below.

**Figure 3a. Alternative 1 Clusters**

<table>
<thead>
<tr>
<th>C1</th>
<th>C2</th>
</tr>
</thead>
<tbody>
<tr>
<td>f v</td>
<td>g o s</td>
</tr>
<tr>
<td>a g p r p</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 3b. Alternative 2 Clusters**

<table>
<thead>
<tr>
<th>C1</th>
</tr>
</thead>
<tbody>
<tr>
<td>f v</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C2'</th>
</tr>
</thead>
<tbody>
<tr>
<td>a g o p r s</td>
</tr>
</tbody>
</table>

**Meta-Rules for Alternative 1:**

- **Figure 3a** is an illustration of the three clusters, namely C1, C2, and C3. Assume that clusters C1, C2, and C3 are searched by processors P1, P2, and P3 respectively, in a 3-node parallel computing environment. The derived meta-rules for controlling the migration of rules, facts, and queries also use the search space and time as follows.

**Meta-Rule1:** if P3 executes the rule(s) pension support or retirement, either (i) migrate the rule(s) old employee and salary from P2 to P3 or (ii) migrate the fact(s) age to P2.

**Meta-Rule2:** if P3 executes the rule(s) fired, nothing is migrated from P1 to P3 or vice versa (except a search-command).

The choice of either option under Meta-Rule1 depends on the amount of data to migrate. However, both meta-rules suggest an opportunity for exploiting further parallelism by merging C2 and C3 into a C2', producing two clusters, C1 and C2'. There is no meta-rule required except a migration of a search-command and an implied cost due to transfer of results. This merging is illustrated by the arrows in Figure 3a.

**Meta-Rules for Alternative 2:**

- **Figure 3b** is an illustration of two clusters, namely D1 and D2, formed from the G2 partitions of Figure 2b. Assume that clusters D1 and D2 are searched by processors P1 and P2; respectively, in a 2-node parallel computing environment. The derived meta-rules that control the migration of rules, facts, and queries also use the search space and time as follows.

26
Meta-Rule3: if N2 executes the rule(s) pension_support or retirement, either (i) migrate the rule(s) old_ employee and salary from N1 to N2 or (ii) migrate the fact(s) age to N1.

Meta-Rule4: if N2 executes the rule(s) fired nothing is migrated from N1 to N2 or vice versa (except a search-command).

Both Meta-Rule3 and Meta-Rule4 suggest the placement of rules old_ employee and salary in cluster D2. The result is now similar to the analysis of the meta-rules under Alternative 1. Thus, the clusters constructed from the G1 and G2 partitions of Figure 2b are equivalent.

The example presented is to motivate the strategies for partitioning KB data for parallel search and reasoning. These methods extend naturally to real, large KB systems. The strategy might be constrained by memory requirements for constructing the matrices, however, the KB may be analyzed a priori and segmented. The clusters in each segment are further analyzed to determine inter-segment dependencies.

The meta-rules and their analysis provides a mechanism for constructing distinct clusters that incur a minimal interprocessor overhead. The example also shows that in spite of the availability, the strategies seek for an optimal use of the processors in the parallel environment.

CONCLUSION

In the paper we have presented a methodology for analyzing, modeling, and partitioning KBs for concurrent search in parallel environments. We have demonstrated the applicability of these strategies through an example, and discussed a mechanism for constructing non-interfering clusters. Non-interfering clusters suggest an opportunity for parallel search with a minimal interprocessor overhead.

REFERENCES


Learning by Analogy on the Connection Machine

Diane J. Cook  
Department of Computer Science  
University of Illinois  
Urbana, IL 61801

ABSTRACT

Analogue planning provides a means of solving problems where other machine learning methods fail, because it does not require numerous examples or a rich domain theory. Instead, analogy utilizes knowledge of solved problems in similar domains, adapting the knowledge to the current problem.

Unfortunately, the analogue planning task is an expensive one. While the process of forming correspondences between a known and a new problem is complex, the problem of selecting a base case for the analogy is virtually intractable. This paper addresses this issue by describing the ANAGRAM planning system, which takes advantage of the massively parallel architecture of the Connection Machine to perform base selection and map formation. This approach makes analogue planning a tractable task, in fact linear in a value less than the size of the plans.

This paper describes the ANAGRAM system and its parallel algorithms. The paper also presents a theoretical analysis and results of testing the system on a large database of plans from the domain of automatic programming.

INTRODUCTION

When solving a problem in a relatively new and unfamiliar domain, a scientist often relies on experience with similar problems to suggest ways of solving the current problem: adapting known techniques, mapping appropriate constraints from a solved problem to the problem at hand, and modifying existing programs to include new capabilities. Analogy is a machine learning technique which can be used to transfer knowledge of a similar domain to the current problem domain. Given a novel problem (the target case), an analogue system selects a similar, solved problem (the base case), computes a mapping between the base and target problem descriptions, and uses the mapping to adapt the base solution to the current domain.

Perhaps the biggest factor that currently prevents machines from making extensive use of learning by analogy is the complexity of the task. Once a base problem has been found and similarities established, construction of a target plan from the known similarities and base solutions is straightforward. However, selecting a base and establishing correspondences are extremely time-consuming processes.

This research addresses the problem of accurately and efficiently forming analogies to solve new problems. The ANAGRAM system solves novel problems by constructing analogue plans using the Connection Machine. Given a target goal, ANAGRAM finds a similar goal in the base domain from which a solution can be derived. ANAGRAM expresses plans as graphs and uses a graph matching algorithm to identify potential base problems and form the mapping between base and target problems. Because many of these tasks take advantage of the massively parallel architecture of the Connection Machine, the process of generating analogue plans in ANAGRAM becomes a tractable task.

This paper presents a means of reducing the time complexity of the analogue planning task by constructing parallel versions of the base selection and map formation algorithms that run on a Connection Machine. Section 2 introduces the ANAGRAM system, a program which generates analogue plans. The next section describes the design of the Connection Machine. The parallel implementations of ANAGRAM's graph match and base selection procedures are detailed in section 4, followed by an example of automatic program generation using the ANAGRAM system. Section 6 presents a complexity analysis of the algorithms. Given enough processors, the parallel analogue planning process is shown to be linear to the size of the base cases. Finally, section 7 presents empirical results using ANAGRAM to generate analogue plans in the domain of automatic programming.

THE ANAGRAM SYSTEM

The ideas mentioned in this paper are implemented in a system called ANAGRAM (ANALogical GRAph Match). Given a target problem specification represented in graph form, ANAGRAM uses a colored graph match technique to select a base case from a database of previously-solved problems. ANAGRAM uses the selected base case to generate a plan which will achieve the target goal.

The system accepts as input two subgraphs, representing the target problem's initial state description and goal state specification. ANAGRAM then searches through the database, finding the best match for both subgraphs. U-
ing the output of the individual graph matches, ANAGRAM then maps over the base plan to the target domain to generate a solution for the target problem.

The analogical planning process is an expensive one when performed sequentially. The following sections describe the parallel implementation of ANAGRAM’s algorithms and illustrate the techniques with an example from the domain of automatic programming. A complexity analysis of the implementation and results of empirical testing are then presented.

THE CONNECTION MACHINE

It is difficult to understand why analogical reasoning is performed so often and so easily by humans, yet is difficult and costly to perform on a machine. Part of the problem is not fully understanding the nature of analogical reasoning and the algorithms humans use to perform it. However, much of the problem is speed. Unfortunately, computers use their powerful hardware inefficiently — most of the transistors in a von Neumann computer are idle at any given instant. Humans make much better use of the brains by computing in parallel. It is only recently that engineers have simulated this approach by building concurrent machines — machines that are able to split a task among various processors which solve the sub-tasks in parallel.

The Connection Machine is a “data parallel” computing system. Most high-performance computers include a small number of fast, relatively complex processors and a single large working-memory space. In contrast, a data-parallel machine has a large number of individual processors, each with a relatively small memory space of its own. The goal is to speed up computation by “parceling out” individual data elements (or small groups or elements) to individual processors, and allowing the processors to operate simultaneously on their own small pieces of the overall problem. This research was implemented on the Connection Machine Model CM-2, built by Thinking Machines Corporation and made accessible through the National Center for Supercomputing Applications. The CM-2 has 32,768 processors structured in a two-dimensional grid. The algorithms described here are implemented in *Lisp, a parallel extension of Common Lisp.

This research makes use of parallel computation in two aspects of analogical plan formation. The graph match algorithm described in the next section is used to examine each base case. The process is implemented in parallel by assigning the contents of each node to a separate processor, and having each node look for a match in the second graph in parallel. In addition, because there are so many processors available, many of the plans stored in the database can be examined at once. Thus the case selection becomes a task with almost constant complexity.

PARALLEL GRAPH MATCH

The graph match algorithm is an essential part of ANAGRAM’s reasoning process. ANAGRAM employs a colored graph match on directed acyclic graphs (DAGs) to determine if the initial and goal states of the target problem match with initial and goal states of a previous problem. If the test is unsuccessful, the list of matched nodes is output. This list is later used to map over the entire base plan to solve the target problem. The links as well as the nodes of the graph are labeled.

The parallel graph match algorithm is an extension of the basic tree isomorphism test presented by (Aho et al. 1974). A level-by-level comparison is done between two graphs $G_1$ and $G_2$. Each node is assigned a_tuple which consists of the level of the node in the graph, the incoming link labels, the outgoing link labels, the integers assigned to parent nodes and to child nodes (described in the next paragraph), and the node label.

Initially each node is assigned the integer -1. As a match is found for a node $n_i$ from $G_1$, $n_i$ and its counterpart $n_j$ from $G_2$ are assigned a unique integer. The integer is propagated to the node’s parents and children, to complete their tuples. Tuples $t_1$ matches tuple $t_2$ if the tuples are complete, the levels match, the incoming links match, the outgoing links match, and the parent and child integers match. If a tuple from $G_1$ is incomplete (the value of a parent or child integer is -1) and there is only one tuple from $G_2$ that could match it, the tuples are matched and added to the list. This match is checked later — if upon completion of the tuples the nodes no longer match, the entire graph match returns failure.

EXAMPLE

Consider the following example from the automatic programming domain. The goal of the target functions is specified as:

$$\text{abs}(a/b - q) < e$$

The goal of the base function currently being examined is:

$$\text{abs}(c/d - q) < i$$

Figure 1 shows the graphs corresponding to the two goal specifications, and the initialized values of the tuples for the two graphs are shown in table 1. The format of a tuple is (level incoming-links outgoing-links parent-ints child-ints NODE LABEL).

In the first pass, each node from the base graph looks for a match from the target graph. No complete matches are found (every tuple has at least one -1), but an incomplete match is found for every tuple. A unique integer $int$ is assigned to the nodes in each unique match. In this case, the integers are assigned from the top to the bottom.
Figure 1: Base and Target Goal Specification

<table>
<thead>
<tr>
<th>Base Goal</th>
<th>Target Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>int&lt;sub&gt;1&lt;/sub&gt;</td>
<td>tuple&lt;sub&gt;1&lt;/sub&gt;</td>
</tr>
<tr>
<td>-1 (1 compare -1 GOAL-STATE)</td>
<td>-1 (1 compare -1 GOAL-STATE)</td>
</tr>
<tr>
<td>-1 (2 compare t1 t2 -1 -1 -1 &lt;)</td>
<td>-1 (2 compare t1 t2 -1 -1 -1 &lt;)</td>
</tr>
<tr>
<td>-1 (3 t1 t2 -1 E)</td>
<td>-1 (3 t1 t2 -1 E)</td>
</tr>
<tr>
<td>-1 (4 arg op -1 -1 TERM0)</td>
<td>-1 (4 arg op -1 -1 TERM0)</td>
</tr>
<tr>
<td>-1 (6 t1 t1 t2 -1 -1 -1 -1)</td>
<td>-1 (6 t1 t1 t2 -1 -1 -1 -1)</td>
</tr>
<tr>
<td>-1 (6 t1 t1 t2 -1 -1 -1 -1)</td>
<td>-1 (6 t1 t1 t2 -1 -1 -1 -1)</td>
</tr>
<tr>
<td>-1 (6 t1 t1 t2 -1 -1 -1 -1)</td>
<td>-1 (6 t1 t1 t2 -1 -1 -1 -1)</td>
</tr>
<tr>
<td>-1 (6 t1 t1 t2 -1 -1 -1 -1)</td>
<td>-1 (6 t1 t1 t2 -1 -1 -1 -1)</td>
</tr>
<tr>
<td>-1 (7 t1 -1 A)</td>
<td>-1 (7 t1 -1 A)</td>
</tr>
<tr>
<td>-1 (7 t1 -1 B)</td>
<td>-1 (7 t1 -1 B)</td>
</tr>
</tbody>
</table>

Table 1: Initial Tuples for Base and Target Goal Subgraphs

<table>
<thead>
<tr>
<th>Base Goal</th>
<th>Target Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>int&lt;sub&gt;1&lt;/sub&gt;</td>
<td>tuple&lt;sub&gt;1&lt;/sub&gt;</td>
</tr>
<tr>
<td>1 (1 compare 2 GOAL-STATE)</td>
<td>1 (1 compare 2 GOAL-STATE)</td>
</tr>
<tr>
<td>2 (3 t1 arg 2 5 ABS)</td>
<td>3 (3 t1 arg 2 5 ABS)</td>
</tr>
<tr>
<td>5 (4 arg op 3 6 TERM0)</td>
<td>5 (4 arg op 3 6 TERM0)</td>
</tr>
<tr>
<td>6 (6 t1 t1 t2 6 9 10 /)</td>
<td>7 (6 t1 t1 t2 6 9 10 /)</td>
</tr>
<tr>
<td>8 (6 t1 t1 t2 6 9 10 /)</td>
<td>9 (6 t1 t1 t2 6 9 10 /)</td>
</tr>
<tr>
<td>9 (7 t1 7 A)</td>
<td>10 (7 t1 7 A)</td>
</tr>
<tr>
<td>10 (7 t1 7 B)</td>
<td>10 (7 t1 7 B)</td>
</tr>
</tbody>
</table>

Table 2: Updated Tuples for Base and Target Goal Subgraphs
of the list starting with int = 1. The integers assigned to each node are shown next to the corresponding node in figure 1. These integers are then propagated to parents and sons through the graph, and corresponding tuples are updated. The values of the tuples and the final matches after updating the tuples are shown in Table 2. The new values are:

- After rechecking the matches, no inconsistencies were found, so the test is completed after the first pass. The mapping that the system outputs is:

\[
\{\text{GOAL-STATE} \rightarrow \text{GOAL-STATE}, \text{c} \rightarrow \text{c}, \text{abs} \rightarrow \text{abs}, \text{e} \rightarrow \text{i}, \text{term}0 \rightarrow \text{term}1, \text{<} \rightarrow \text{>,} / \rightarrow \text{/,} \text{a} \rightarrow \text{e}, \text{b} \rightarrow \text{d}\}
\]

**COMPLEXITY ANALYSIS**

In the parallel implementation, each node in graph \(G_1\) simultaneously searches for a match. After a single search pass is completed, the tuples in \(G_1\) update their information in parallel. The complexity of each search pass is \(O(1)\). If the arc labels are not sufficiently constraining to allow only one possible match for each node, more than one pass must be made. Before the next pass, the algorithm selects a single node that has more than one possible match, and randomly selects a match from the list of candidates (this does not affect the chances of finding a complete map between the two graphs). Let \(n\) be the number of nodes in both \(G_1\) and \(G_2\), and let \(h\) be the height of both graphs. Note that the height is the length of the longest path from any leaf to any root in the graph. The number of passes needed to complete the graph match is \(n - h\), thus the complexity of the parallel graph match algorithm is \(O(n - h)\).

Parallelizing the graph match algorithm greatly reduces the complexity of the task (the complexity of the sequential algorithm for testing the isomorphism of two graphs is \(O(n^3)\)). An even greater speedup in the analogical process occurs when base selection is parallelized. The plans found in the database are all stored in separate processor subsets. The number of plans stored depends on the size of the graphs and the number of processors available in the machine. Plans can be stored in groups according to their key-word indices. The size of each group is determined by the number that can be searched at once. The goal subgraphs of each plan are matched in parallel, and the results of these matches are used to match the initial state subgraphs. The complexity of the base selection process is thus \(2 \times (n - h)\), or \(O(n - h)\).

**EMPIRICAL RESULTS**

This section describes a series of experiments designed to test the performance of the parallelized graph match algorithm and base selection process. The first experiment tests the graph match algorithm, and the second experiments tests the base selection algorithm.

**Experiment 1**

In order to empirically judge the performance of ANAGRAM's graph match algorithm, the following experiment was designed. A Lisp function was created to generate code specifications of various sizes. To show how performance is affected as the size of the graph increases, graphs were generated of size 4 through size 200 (usually by increments of 10). Noise was minimized by generating 50 graphs of each desired size.

Figure 2 shows the results of this experiment. The x axis represents the size of the two graphs being matched (size \(n\) means there are \(n\) nodes in each of the two graphs). The y axis represents the time taken to match the two graphs, in numbers of CPU seconds.

Three variations of the algorithm were tested. The first curve shows the sequential version of the algorithm implemented on a TI Explorer II. The second curve shows the results of the parallel implementation, run on a Connection Machine II. The third curve represents the results of the sequential implementation run on the Connection Machine II. The sequential algorithm runs on a single processor on the Connection Machine. This last curve was included to more accurately compare the results of the TI tests with the Connection Machine results.

The results shown in the graph come very close to expectation. The complexity of the sequential algorithm was theoretically shown to be \(O(n^2)\) in the worst case. The average case complexity appears to be slightly better, though definitely more costly than a linear function.

On the other hand, the complexity of the parallel algorithm was given as \(O(n - h)\), which would make the results better than linear. The graph shows that the results are slightly better than linear, but very close to linear. This is not surprising, because each of the processors on the Connection Machine is very small, and a small increase in the amount of information contained in each node can contribute to a large decrease in CPU speed.

As is demonstrated in the graph by the CM Sequential curve, a single processor on the Connection Machine is much less powerful than the single processor found on the TI Explorer II. Thus it can be assumed that if each processor on the Connection Machine were as powerful as a TI II, not only would the entire CM Parallel curve be shifted negatively in the y direction, but the point at which the curve flattens out which occur much sooner.

**Experiment 2**

The second experiment is designed to compare the results of the base selection algorithm as performed on the TI II and on the CM-II. Using the graph-generator described in experiment 1, 500 sets of graphs of size 20 were generated. Each set consists of a goal-state subgraph of size 20, and a initial-state subgraph of size 20. The experiment ran
the base selection algorithm on a database of \( n \) graphs. For each test, the database was designed such that the only successful candidate base case would be the last one examined.

Figure 3 illustrates the results of the experiment. The markers on both curves show the data points collected. The increments between data points on the Connection Machine curve are larger, because of the time spent in setting up the database on the Connection Machine. It is important to note that converting graphs to the representation used on the CM-II is a very cpu-expensive process. Because this conversion is not part of the base selection process (in theory the database is updated as new plans are entered into the system), the corresponding CPU time was subtracted out of the results. As would be expected, the TI Base Selection curve shows that the amount of time spent in base selection increases linearly in the size of the database. In contrast, the CM Base Selection curve is almost constant. The slight increase in time on this second curve reflects the increased amount of overhead involved in checking to see when a match is completed. It also reflects the overhead inherent in increasing the amount of information contained in each node.

**CONCLUSION**

Learning by analogy is an effective means of solving problems in new domains given knowledge about problems in similar domains. Unfortunately, the complexity of the task has made developing an analogical reasoning system intractable. This research presents a graph-matching approach to analogical reasoning that is implemented on the Connection Machine. The results of theoretical and empirical analysis show that the analogical planning process is a tractable one when the process can be parallelized. With these results, ANAGRAM can now be used to exploit the power of analogical reasoning unencumbered by the nemesis of computational complexity.

**ACKNOWLEDGEMENTS**

I would like to thank Robert Stepp for his helpful suggestions. I would also like to thank the National Center for Supercomputing Applications for the use of the Connection Machine.

**REFERENCES**

KNOWLEDGE ENGINEERING A PARALLEL FORWARD CHAINING INFERENCING
SYSTEM

Claudia P. Industrious, Lawrence O. Hall and Onzik Kim

Department of Computer Science and Engineering
University of South Florida
Tampa, Fl. 33620
e-mail: hall@usf.edu

ABSTRACT

An inferencing mechanism is a fundamental component in rule-based expert systems. It manipulates the rules and facts in the knowledge base and working memory to effect reasoning within the system. In recent times there have been a number of research efforts into the feasibility of increasing the computational speed of the inferencing procedure in rule-based systems via parallel programming techniques. This paper presents the results from experimentation with an experimental model of a forward chaining inferencing mechanism designed for execution in a parallel processing environment. The experiments described here were performed on a 10 processor Encore Multimax computer, a shared memory architecture machine.

INTRODUCTION

PHFC (Parallel Hierarchal Forward Chaining system) is the name given to an inferencing methodology designed to execute in a parallel processing environment. The details of the methodology can be found in (Krall, et. al 1986) and will only be described briefly here. Datapac is the name given to an implementation of the PHFC scheme. It includes a collection of rule sets with varying characteristics. The results of experiments performed in a simulated environment can also be found in (Krall, et. al 1986).

The PHFC approach attempts to exploit parallelism in rule inferencing by formulating a hierarchal structure based on precedence (i.e. predecessor-successor) relationships between rules. The structure is then used to effect implicit control of the inference procedure for rules in the set.

The general format of rules in the system is as follows:

(level-number rule-id premise conclusion)

The level number of a rule is a numerical representation of the number of rule groups in the hierarchy for which inferencing must be performed before inferencing for the given rule can commence. That is, the depth in the hierarchy of rules in which it is situated. It is determined based on conditions specified in its premise. The rule-id (rule identifier) is used for specifying particular rules within the system. The premise specifies the conditions that must be met before the rule can fire. It consists of a conjunction or disjunction of any number of clauses in any depth. The conclusion specifies the facts that are to be asserted into working memory, when the premise is deemed to be true.

THE UNDERLYING ARCHITECTURE

The experimentation and results described in this paper were executed on an Encore Multimax multiprocessor computer equipped with 10 processors. The Encore Multimax has a shared-memory architecture. All communication between processors, I/O devices, and memories utilizes utilizes a high-speed system bus (Nanobus). Each processor is capable of executing .75 million instructions per second, and has a 64K cache associated with it.

MODIFICATIONS TO DATAPAC - ANALYSIS AND RESULTS

Experimentation with the PHFC system has been performed in CSIM (C-based simulation) Multilisp and on an Encore multiprocessor equipped with 10 processors. The results presented here are from executions on the Encore multiprocessor.

They can be summarized in terms of speedups, which measure how well the scheme benefits from execution on multiple processors. Speedup is defined to be the ratio of elapsed time required by the sequential (one-processor) version of the system to the elapsed time of a N-processor version. The calculations are shown in the tables at the end of this document. The results indicate that improvements in terms of speedup can be realized with this method. For example, there was a speedup of 5.6 with 10 processors. Since the speedups realized with a given data set in the PHFC scheme are related to its structure, The
pseudo maximum parallelism (PMP) which is a measure of the average number of rules in a level is also of importance. For this reason the ratio of speedup to the average number of rules in the rule sets (SP/RL) is also presented in the tables. Seven knowledge bases were used as the basis of experimentation in the original version of Datapac. The general strategy for accessing the worth of the modifications described here was to first perform experiments with a single knowledge base (KB), and after it was determined that major improvements could be made with the change, experimentation was performed with other knowledge bases.

**Modification 1: Premature Termination of Non-or Rule Processors**

The first modification to Datapac to be discussed here attempted to exploit the hierarchal relationship between rules by terminating processors of non-or rules (a non-or rule is one whose premise is a conjunction of clauses, and will therefore only fire if all its predecessors have) prematurely before going through the inferencing phase whenever one of their predecessors failed to fire.

This modification had little effect on the execution time or speedup figures of the original version of the system (See Tables 1, 2 and 5, 6). A possible explanation for the behavior observed here is that the amount of work performed by non-or rule processors in this case is not enough to justify the overhead incurred by the creation of additional processes (with this approach each rule inference constitutes a process). Because the overhead associated with the creation of tasks in a parallel processing environment, it is not desirable to create tasks to perform the computation for small problem instances. This problem can probably be alleviated by assigning the inferencing of entire predecessor-successor chains to processors instead of assigning the inferencing of single rules to each one. The effect on system performance with this change is dependent upon the level depth of the KBs.

**Modification 2: Minimized Number of Active Processes**

In Datapac every rule is assigned to a processor at the beginning of the inferencing phase. Although there is no premature inferencing, there is still a significant amount of overhead incurred due to the queuing of tasks normally associated with scheduling and with semaphores. In addition, with the PHFC scheme there is intentional blocking of rule processes whenever precedence conditions for the corresponding rules have not been examined. A second modification to the system attempted to minimize the number of active processes in the system to approximately n, where n is the number of available processors. In this version of Datapac, the total number of available cpus is determined at initialization time. Access to the count is made on an exclusive basis only. A central process (dispatch) monitors the availability of processors in the system by periodically checking the value of the count. When a processor becomes available (count ≥ 1), it spawns a rule process by removing a rule id from the list of rule ids created during preprocessing and creating a rule engine process for the given rule. It then decrements the count. Whenever a rule process completes the inferencing procedure for the assigned rule, it increments the processor count. As can be seen from the results in Tables 7 thru 9 there is not a significant difference in speedup with this modification. The noteworthy aspect of this scheme is that there is a 50 percent reduction in execution time. The 50 percent reduction in execution time can be attributed to several factors. In the original version of the program the status flag used to indicate that inferencing on a rule was complete was set twice. The flag values are stored as array elements and are set by each rule process. A portion of the reduction in execution time is due to the removal of the duplicate code (in Lisp, the manipulation of array elements can be expensive). Further, in the original version, rule processes were initiated by applying the Lisp MAP-CAR function to a list of rule identifiers in the knowledge base. This technique for creating rule inference processes was removed in order to better control the number of active processes in the system at any given time.

**Modification 3: Minimized Contention**

With bus-based shared memory architectures two problems that frequently arise in attempts to obtain the maximum efficiency and speed in applications are the problems of contention for the same memory location and the rising cost of overall speed gain (i.e. there is an increase from one to about five or six processors then after that adding processors fails to increase speed). In general the number of processors used should be less than some maximum based on the granularity of tasks and the processing cost. Since contention is a major concern in these systems, it is useful to identify and remove (if possible) any areas in the inferencing scheme that contribute to it. In the PHFC scheme a post office working memory mechanism is shared by all processes and is a major contributor to contention in the system. An attempt was made to remove this mechanism but with undesirable results (See Table 10).

There are several possible explanations for the difference in what may initially be expected and the actual performance. Intermediate as well as final results are reported by the system. The post office working memory mechanism provides a convenient place to store intermediate results to be reported at the end of inferencing. The alternatives are (1) random broadcasting of results as they are determined within the system; or (2) copying intermediate results of a given rule inferencing in the working memories of rule processors at lower levels, so that the final result of the rule inferencing procedure may be reported from the working memories of rule processors at the bottom level (which will include all intermediate results since they would have propagated down to the bottom level working memories). The first alternative is clearly an unacceptable one. The second alternative, even though more acceptable, requires
more writes (and hence additional overhead with synchronization requirements) than the original system. This is undesirable also.

SUMMARY AND CONCLUSIONS

The PHFC mechanism is inherently a modular system. The modularity in this methodology arises from the fact that sets of rules (essentially predecessor-successor chains) can be processed as functional units requiring little communication with other units in the system. Rules in a unit/set are inter-related (they are predecessors and successors of each other), and the PHFC scheme allows inferencing to be performed for only one rule in a set at any given time. This suggests that a chunking mechanism that assigns groups of rules in predecessor-successor chains to processors would probably be beneficial. The predecessor-successor sets can be determined at preprocessing time. If this mechanism were implemented, one benefit would be that there would no longer be any need for rule processors to intentionally block themselves when the examination of precedence conditions have not been completed. The fact some rules will appear in more than one chain will not pose a major problem since the status flag facility can be used for verifying that inference for a given rule has been performed; so that there will be no redundant computations for these rules.

Using placeholders (which in Multilisp could be seen as an implementation of spin locks) as a means of synchronization has been suggested as a means of reducing the overhead associated with semaphores. There has been some experimentation with this possibility. It was found that placeholders can be used as a means of synchronization between processes, however, time did not permit for the completion of testing of placeholders in Datapac.

Another possibility for improved performance in the Datapac implementation is the introduction of a filtering mechanism which reduces the number of rules involved in a given exchange with the system based on knowledge of its requirements. This change will essentially involve the creation of a pruned set of rules at preprocessing time that includes only those rules that are relevant to the given exchange. The reduction of the number of rules that need to be considered during the inferencing procedure for a given procedure will result in a reduction in the overall processing time for inferencing. There are tradeoffs with this method; however, mainly the additional overhead at preprocessing time for creating the streamlined set of rules. There is also the question of where to get the knowledge about what can be immediately pruned (an old and difficult problem).

The scheme presented here is simple and works only for a limited problem set. Investigation into scaling it for OPS type problems remains to be done. It shows performance gains on a real machine with some small knowledge bases. The paradigm may be one technique to help improve expert system performance with large knowledge bases.

ACKNOWLEDGEMENTS:

Research partially supported by NASA-AMES grant NAG-2-487 and a NASA minority supplement grant.

Thanks to Encore for the use of their parallel computer.

REFERENCES


Results of original PHFC scheme

<table>
<thead>
<tr>
<th>Sequential Execution</th>
<th>6.71 seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parallel Execution:</td>
<td></td>
</tr>
<tr>
<td>Number of Execution</td>
<td>Speedup</td>
</tr>
<tr>
<td>Speedup</td>
<td></td>
</tr>
<tr>
<td>processors</td>
<td>Time (SP)</td>
</tr>
<tr>
<td>1</td>
<td>7.30</td>
</tr>
<tr>
<td>2</td>
<td>3.58</td>
</tr>
<tr>
<td>3</td>
<td>2.80</td>
</tr>
<tr>
<td>4</td>
<td>2.80</td>
</tr>
<tr>
<td>5</td>
<td>1.81</td>
</tr>
<tr>
<td>6</td>
<td>1.98</td>
</tr>
<tr>
<td>7</td>
<td>1.36</td>
</tr>
<tr>
<td>8</td>
<td>1.36</td>
</tr>
<tr>
<td>9</td>
<td>1.36</td>
</tr>
<tr>
<td>10</td>
<td>1.31</td>
</tr>
</tbody>
</table>

*RL = average number of rules per level = 15
Number of levels = 6
Table 1: Speedup Evaluation with 90 Non-variable knowledge base.
# Results of original PHFC scheme

## Sequential Execution: 7.38 seconds

### Parallel Execution:

<table>
<thead>
<tr>
<th>Number of processors</th>
<th>Execution Time (SP)</th>
<th>Speedup (SP/RL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7.65</td>
<td>.96</td>
</tr>
<tr>
<td>2</td>
<td>4.05</td>
<td>1.82</td>
</tr>
<tr>
<td>3</td>
<td>2.80</td>
<td>2.63</td>
</tr>
<tr>
<td>4</td>
<td>2.10</td>
<td>3.51</td>
</tr>
<tr>
<td>5</td>
<td>1.75</td>
<td>4.21</td>
</tr>
<tr>
<td>6</td>
<td>1.98</td>
<td>3.72</td>
</tr>
<tr>
<td>7</td>
<td>1.36</td>
<td>5.42</td>
</tr>
<tr>
<td>8</td>
<td>1.18</td>
<td>6.25</td>
</tr>
<tr>
<td>9</td>
<td>1.33</td>
<td>5.54</td>
</tr>
<tr>
<td>10</td>
<td>1.31</td>
<td>5.63</td>
</tr>
</tbody>
</table>

*RL = average number of rules per level = 30.5

## Number of levels = 2

Table 2: Speedup Evaluation with Fevers Knowledge Base

## Results of original PHFC scheme

## Sequential Execution: 9.47 seconds

### Parallel Execution:

<table>
<thead>
<tr>
<th>Number of processors</th>
<th>Execution Time (SP)</th>
<th>Speedup (SP/RL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11.20</td>
<td>.94</td>
</tr>
<tr>
<td>2</td>
<td>4.78</td>
<td>1.98</td>
</tr>
<tr>
<td>3</td>
<td>3.20</td>
<td>2.95</td>
</tr>
<tr>
<td>4</td>
<td>2.83</td>
<td>3.34</td>
</tr>
<tr>
<td>5</td>
<td>2.85</td>
<td>3.32</td>
</tr>
<tr>
<td>6</td>
<td>2.50</td>
<td>3.78</td>
</tr>
<tr>
<td>7</td>
<td>2.03</td>
<td>4.66</td>
</tr>
<tr>
<td>8</td>
<td>1.96</td>
<td>4.83</td>
</tr>
<tr>
<td>9</td>
<td>2.00</td>
<td>4.73</td>
</tr>
<tr>
<td>10</td>
<td>2.15</td>
<td>4.40</td>
</tr>
</tbody>
</table>

*RL = average number of rules per level = 5

## Number of levels = 5

Table 3: Speedup Evaluation with 25 Variable Rule Knowledge Base

## Results of original PHFC scheme

## Sequential Execution: 89.52 seconds

### Parallel Execution:

<table>
<thead>
<tr>
<th>Number of processors</th>
<th>Execution Time (SP)</th>
<th>Speedup (SP/RL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>132.70</td>
<td>.67</td>
</tr>
<tr>
<td>2</td>
<td>50.36</td>
<td>1.77</td>
</tr>
<tr>
<td>3</td>
<td>35.11</td>
<td>2.54</td>
</tr>
<tr>
<td>4</td>
<td>29.58</td>
<td>3.02</td>
</tr>
<tr>
<td>5</td>
<td>23.08</td>
<td>3.87</td>
</tr>
<tr>
<td>6</td>
<td>20.63</td>
<td>4.33</td>
</tr>
<tr>
<td>7</td>
<td>19.80</td>
<td>4.56</td>
</tr>
<tr>
<td>8</td>
<td>19.60</td>
<td>4.56</td>
</tr>
<tr>
<td>9</td>
<td>12.85</td>
<td>6.96</td>
</tr>
<tr>
<td>10</td>
<td>12.58</td>
<td>7.11</td>
</tr>
</tbody>
</table>

*RL = average number of rules per level = 30.5

## Number of levels = 2

Table 4: Speedup Evaluation with 100 Variable Rule Knowledge Base

## Mod for early termination of "Non-or" rules

## Sequential Execution: 6.71 seconds

### Parallel Execution:

<table>
<thead>
<tr>
<th>Number of processors</th>
<th>Execution Time (SP)</th>
<th>Speedup (SP/RL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7.05</td>
<td>.95</td>
</tr>
<tr>
<td>2</td>
<td>3.60</td>
<td>1.86</td>
</tr>
<tr>
<td>3</td>
<td>2.61</td>
<td>2.54</td>
</tr>
<tr>
<td>4</td>
<td>3.30</td>
<td>2.03</td>
</tr>
<tr>
<td>5</td>
<td>1.66</td>
<td>4.02</td>
</tr>
<tr>
<td>6</td>
<td>1.85</td>
<td>3.62</td>
</tr>
<tr>
<td>7</td>
<td>1.35</td>
<td>4.90</td>
</tr>
<tr>
<td>8</td>
<td>1.36</td>
<td>4.91</td>
</tr>
<tr>
<td>9</td>
<td>1.28</td>
<td>5.24</td>
</tr>
<tr>
<td>10</td>
<td>1.28</td>
<td>5.22</td>
</tr>
</tbody>
</table>

*RL = average number of rules per level = 15

## Number of levels = 6

Table 5: Speedup Evaluation with 90 Non-variable Knowledge Base

## Mod for early termination of "Non-or" rules

## Sequential Execution: 7.38 seconds

### Parallel Execution:

<table>
<thead>
<tr>
<th>Number of processors</th>
<th>Execution Time (SP)</th>
<th>Speedup (SP/RL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8.20</td>
<td>.90</td>
</tr>
<tr>
<td>2</td>
<td>4.28</td>
<td>1.72</td>
</tr>
<tr>
<td>3</td>
<td>2.95</td>
<td>2.48</td>
</tr>
<tr>
<td>4</td>
<td>2.76</td>
<td>2.66</td>
</tr>
<tr>
<td>5</td>
<td>1.98</td>
<td>3.72</td>
</tr>
<tr>
<td>6</td>
<td>1.73</td>
<td>4.25</td>
</tr>
<tr>
<td>7</td>
<td>1.55</td>
<td>4.76</td>
</tr>
<tr>
<td>8</td>
<td>1.51</td>
<td>4.86</td>
</tr>
<tr>
<td>9</td>
<td>1.36</td>
<td>5.40</td>
</tr>
<tr>
<td>10</td>
<td>1.43</td>
<td>5.15</td>
</tr>
</tbody>
</table>

*RL = average number of rules per level = 30.5

## Number of levels = 2

Table 6: Speedup Evaluation with Fevers Knowledge Base
### Modification for limits on number of processes

**Sequential Execution:** 4.05 seconds

<table>
<thead>
<tr>
<th>Number of processors</th>
<th>Execution Time</th>
<th>Speedup (SP)</th>
<th>Speedup (SP/RL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4.41</td>
<td>3.91</td>
<td>.26</td>
</tr>
<tr>
<td>2</td>
<td>2.51</td>
<td>1.61</td>
<td>.11</td>
</tr>
<tr>
<td>3</td>
<td>1.71</td>
<td>2.36</td>
<td>.16</td>
</tr>
<tr>
<td>4</td>
<td>1.35</td>
<td>3.00</td>
<td>.20</td>
</tr>
<tr>
<td>5</td>
<td>1.11</td>
<td>3.64</td>
<td>.24</td>
</tr>
<tr>
<td>6</td>
<td>1.03</td>
<td>3.93</td>
<td>.26</td>
</tr>
<tr>
<td>7</td>
<td>.93</td>
<td>4.35</td>
<td>.29</td>
</tr>
<tr>
<td>8</td>
<td>.92</td>
<td>4.40</td>
<td>.29</td>
</tr>
<tr>
<td>9</td>
<td>.86</td>
<td>4.76</td>
<td>.32</td>
</tr>
<tr>
<td>10</td>
<td>.83</td>
<td>4.87</td>
<td>.32</td>
</tr>
</tbody>
</table>

*RL = average number of rules per level = 15
Number of levels = 8

Table 7: Speedup Evaluation with 90 Non-variable Knowledge Base

**Parallel Execution:**

<table>
<thead>
<tr>
<th>Number of processors</th>
<th>Execution Time</th>
<th>Speedup (SP)</th>
<th>Speedup (SP/RL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4.80</td>
<td>.96</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2.41</td>
<td>1.91</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2.08</td>
<td>2.21</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1.70</td>
<td>2.71</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1.61</td>
<td>2.86</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1.60</td>
<td>2.88</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1.51</td>
<td>3.05</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1.50</td>
<td>3.07</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1.43</td>
<td>3.22</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>1.60</td>
<td>2.88</td>
<td></td>
</tr>
</tbody>
</table>

*RL = average number of rules per level = 5
Number of levels = 5

Table 8: Speedup Evaluation with 25 Variable Rule Knowledge Base

### Version without post office working memory

**Sequential Execution:** 5.71 seconds

<table>
<thead>
<tr>
<th>Number of processors</th>
<th>Execution Time</th>
<th>Speedup (SP)</th>
<th>Speedup (SP/RL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10.90</td>
<td>.61</td>
<td>.04</td>
</tr>
<tr>
<td>2</td>
<td>5.51</td>
<td>1.21</td>
<td>.08</td>
</tr>
<tr>
<td>3</td>
<td>3.76</td>
<td>1.78</td>
<td>.12</td>
</tr>
<tr>
<td>4</td>
<td>3.35</td>
<td>2.00</td>
<td>.13</td>
</tr>
<tr>
<td>5</td>
<td>2.61</td>
<td>2.56</td>
<td>.17</td>
</tr>
<tr>
<td>6</td>
<td>2.68</td>
<td>2.50</td>
<td>.17</td>
</tr>
<tr>
<td>7</td>
<td>2.06</td>
<td>3.24</td>
<td>.22</td>
</tr>
<tr>
<td>8</td>
<td>1.93</td>
<td>3.47</td>
<td>.23</td>
</tr>
<tr>
<td>9</td>
<td>1.71</td>
<td>3.51</td>
<td>.26</td>
</tr>
<tr>
<td>10</td>
<td>2.18</td>
<td>3.07</td>
<td>.20</td>
</tr>
</tbody>
</table>

*RL = average number of rules per level = 15
Number of levels = 6

Table 9: Speedup Evaluation with 90 Non-variable rule Knowledge Base

### Modification for limits on number of processes

**Sequential Execution:** 3.56 seconds

<table>
<thead>
<tr>
<th>Number of processors</th>
<th>Execution Time</th>
<th>Speedup (SP)</th>
<th>Speedup (SP/RL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4.18</td>
<td>.85</td>
<td>.03</td>
</tr>
<tr>
<td>2</td>
<td>2.28</td>
<td>1.56</td>
<td>.05</td>
</tr>
<tr>
<td>3</td>
<td>1.71</td>
<td>2.08</td>
<td>.07</td>
</tr>
<tr>
<td>4</td>
<td>1.25</td>
<td>2.84</td>
<td>.09</td>
</tr>
<tr>
<td>5</td>
<td>1.05</td>
<td>3.39</td>
<td>.11</td>
</tr>
<tr>
<td>6</td>
<td>0.88</td>
<td>4.05</td>
<td>.13</td>
</tr>
<tr>
<td>7</td>
<td>0.78</td>
<td>4.56</td>
<td>.15</td>
</tr>
<tr>
<td>8</td>
<td>0.83</td>
<td>4.28</td>
<td>.14</td>
</tr>
<tr>
<td>9</td>
<td>0.70</td>
<td>5.08</td>
<td>.17</td>
</tr>
<tr>
<td>10</td>
<td>0.68</td>
<td>5.23</td>
<td>.17</td>
</tr>
</tbody>
</table>

*RL = average number of rules per level = 30.5
Number of levels = 2

Table 8: Speedup Evaluation with Fevers Knowledge Base

**Parallel Execution:**

<table>
<thead>
<tr>
<th>Number of processors</th>
<th>Execution Time</th>
<th>Speedup (SP)</th>
<th>Speedup (SP/RL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10.90</td>
<td>.61</td>
<td>.04</td>
</tr>
<tr>
<td>2</td>
<td>5.51</td>
<td>1.21</td>
<td>.08</td>
</tr>
<tr>
<td>3</td>
<td>3.76</td>
<td>1.78</td>
<td>.12</td>
</tr>
<tr>
<td>4</td>
<td>3.35</td>
<td>2.00</td>
<td>.13</td>
</tr>
<tr>
<td>5</td>
<td>2.61</td>
<td>2.56</td>
<td>.17</td>
</tr>
<tr>
<td>6</td>
<td>2.68</td>
<td>2.50</td>
<td>.17</td>
</tr>
<tr>
<td>7</td>
<td>2.06</td>
<td>3.24</td>
<td>.22</td>
</tr>
<tr>
<td>8</td>
<td>1.93</td>
<td>3.47</td>
<td>.23</td>
</tr>
<tr>
<td>9</td>
<td>1.71</td>
<td>3.51</td>
<td>.26</td>
</tr>
<tr>
<td>10</td>
<td>2.18</td>
<td>3.07</td>
<td>.20</td>
</tr>
</tbody>
</table>

*RL = average number of rules per level = 15
Number of levels = 6

Table 10: Speed-up Evaluation with 90 Non-variable rule Knowledge Base
Adversary Modeling with Multiple Knowledge Sources

Steven Walczak
Computer and Information Sciences Dept.
University of Florida
Gainesville, Florida 32611
email: smur@beach.cis.ufl.edu

Abstract

Artificial intelligence programs operating in tactical and strategic planning domains must account for an adversary's actions. A method of using information from multiple knowledge sources to model an adversary is presented. The adversary modeling method is implemented in the program IAM for the domain of chess. Each knowledge source used attempts to cover a different aspect of the factors that affect the strategic decisions made by an adversary.

INTRODUCTION

Artificial intelligence programs operating in tactical and strategic planning domains must account for the possible counter-plans of an adversary. Not considering the actions of an adversary when making strategic and tactical plans is analogous to creating a chess program which only searches to a depth of one ply or half-move in the program's tree of allowable moves. The creation of a sub-optimal chess program is a ridiculous proposition, however many current military tactical planners such as TACPLAN (Andriole 1986) do not account for the counter-moves of an adversary. Additional tactical planners like ARES (Young and Lehter 1988) prescribe a static number of predefined moves that an adversary can execute. The technique of allowing only a static number of predefined moves is analogous to a chess program that only considers the movement of pawns by an adversary which is also an unrealistic proposition. A few political, POLITICS (Carbonell 1981), and military planners, MARK I and MARK II (Davis 1988), try to model adversaries by defining goals that the adversary will attempt to accomplish. Using goals for adversaries incorporates knowledge into the strategic planners based on the programmer's belief of what an adversary is likely to attempt. This knowledge source can often be misleading or an incorrect evaluation of an opponent's strategies. The bombing of Pearl Harbor by the Japanese during World War II and the Tet Offensive in the Vietnam War are historical reminders of the strategic blindness that occurs when a military strategist assumes the intentions and strategies of an adversary are the same as his own. Additionally, the use of predefined goals to model an adversary's strategy does not permit the model to be adaptive to changes in strategy that an adversary can make.

This paper will detail an adversary modeling method which overcomes the shortcomings of previous methods by utilizing multiple reliable knowledge sources. The knowledge sources each supply information about a different factor (see figure 1) that affects the strategic decision making process of an adversary. The program IAM, Inductive Adversary Modeler, implements the modeling method in the domain of chess. Chess was chosen because of the combination of tactical and strategic elements present in the game and a level of complexity that prohibits a solution from being found by exhaustive search.

Figure 1: Factors affecting an adversary's strategic decisions.

PRIMARY KNOWLEDGE SOURCE

The first obstacle to be overcome is acquiring a reliable knowledge source to form the foundation of the model.
that will prevent strategic blindness from occurring. Waterman’s draw poker program (Waterman 1970) analyzed the playing style of its adversary to develop betting strategies. IAM expands on Waterman’s design by analyzing the performance of an adversary in previous chess games played by the adversary. Alekhine, Botvinnik, and most chess masters study the games of an opponent prior to playing against that opponent (Schonberg 1973) to detect the strengths and weaknesses of play and gain familiarity with their adversary’s style of play. The study of an opponent’s prior games by human chess masters supports the analysis of previous games of an adversary to capture the strategic tendencies of an adversary.

Records of an adversary’s previous games, including the individual moves of each player in algebraic notation and the outcome of the game for the adversary, are used as input to IAM. Chase and Simon have shown that chess masters store from 10,000 to 100,000 patterns of chess pieces (Chase and Simon 1988). These patterns frequently have scripts or plans associated with them that the masters use to respond to specific board configurations. IAM uses the results of Chase and Simon to hypothesize that chess masters will attempt to maneuver board positions into positions that contain familiar patterns. Induction is performed on the previous games of an adversary to recognize recurring move sequences and patterns of pieces. The patterns of pieces discovered by the induction mechanism must be a result of the strategy of the adversary. The requirement for strategic knowledge to be contained in the patterns implies that the induced patterns of pieces must contain a minimum of two pieces, so as not to result from chance occurrence, and at least one piece must belong to the adversary being analyzed. The recurring patterns of pieces and move sequences are identified as part of the collection of patterns known by the chess master and are used to predict the tactical and strategic decisions an adversary will make when analogous situations occur in the future. The adversary model predicts adversary tactics by finding board positions that contain most of the pieces of an induced pattern in their proper locations and the remaining pieces of the pattern can be moved into their proper locations in one or two moves. Based on the hypothesis stated above, an adversary will make the moves that will produce the pattern of pieces contained in the adversary model.

Nine games from the 1948 Hague-Moscow World Championship Tournament played by Botvinnik, taken from (Gelo 1988), were analyzed by IAM. Two of the eighteen chunks found for these nine games are displayed in figure 2. The next game in the tournament shown in figure 3, played by Botvinnik against Keres, was used to test the adversary model. The model detected the possibility for two patterns contained in the model, shown in figure 2, to be reproduced in the current game. The adversary model predicted that the first pattern would cause Botvinnik to castle to the king side of the board on either move thirteen or move fifteen. As seen in figure 3, Botvinnik made the predicted move for his fifteenth game turn, the first move following the second detection of an analogous board position by the adversary model. The second pattern shown in figure 2 was used to predict that Botvinnik would move his knight from d3 to either b4 or e5, with equal probability since both moves reproduce the same pattern, on the twenty-first move of the game. Again, as seen in the game record shown in figure 3 Botvinnik executed the move predicted by the adversary model. The model is reliable because the knowledge source used is the direct observation of the tactics and strategies employed by an adversary. The use of previous actions to predict future actions has been validated by several psychological experiments including the water jug problem (Anderson 1980) and the 1023-choice task (Laird et al. 1988). The mentioned experiments demonstrated that once subjects, in the experiments, acquired a strategy for solving the problems, they will continue to use the acquired strategy even though a less difficult strategy would also solve a problem.

The adversary model is responsive to changes in an adversary’s strategy. Patterns induced from more recent games will be weighted such that these more recent patterns will have a greater influence on the predictions made by the model of an adversary’s probable strategy and tactics.

The performance of tactical and strategic planning programs will be enhanced when adversary models are implemented. The model described thus far suffers from the limitation of only being applicable in tactical and strategic situations that are analogous to the patterns found by the induction mechanism. This limitation arises because the model of an adversary is constructed from a single knowledge source, the record of previous games played by the adversary. The performance of a model increases as additional knowledge about the problem domain is added to the model (Lenat and Peigenbaum 1987). Additional sources of knowledge must be added to the adversary model to increase the model’s performance and applicability.

**ADDITIONAL KNOWLEDGE SOURCES**

The current knowledge source used by IAM is specific to the domain experience of particular adversaries. Knowledge which pertains to a broad class of adversaries is available as another knowledge source. People have unique behavioral patterns due to their social heritage and educational background (Bond 1986). Knowledge about an adversary’s culture and education can be exploited to
Botvinnik v. Keres (Moscow, May 4/5, 1948)
4th Series, 20th Round
Queen’s Gambit Declined

1. d4 d5
2. Nf3 Bf5
3. c4 e6
4. c×d5 e×d5
5. Qb3 Nc6
6. Bg5 Be7
7. B×e7 Ng×e7
8. e3. Qd6
9. Nbd2 0-0
10. Rc1 a5
11. a3 Rc8
12. Bd3 a4
13. Qc2 B×d3
14. Q×d3 Nd8
15. 0-0 Ne6
16. Rc3 (see diagram)
16. . . . . . . . . . .
17. Qc2 Rcb8
18. Ne1 Ncb8
19. Rcb6 Q×b6
20. Nd3 Nb6
21. Nf4 Rd8
22. Qf5 Rd6
23. Rfc1 R×c6
24. R×c6 Rd8

After 16. Rc3

37. Nb4+ Kd6
38. e4 d×e4+
39. f×e4 Ne6
40. Ke3 Nc7
41. Kd3 Ne6
42. Nd5 Kc6
43. h4 Nd8
44. Nf4 Kd6
45. Nh5 Ne6
46. Ke3 Ke7
47. d5 Nc5
48. N×g7 Kd6
49. Ne6 Nd7

Figure 3: Game record of Botvinnik-Keres game used to test IAM (Gelo 1988).

provide the adversary model with a wider range of situations in which probable adversary actions can be predicted. General Robert E. Lee of the Confederate States Army, who is considered by many historians to be one of the greatest military strategists (Davis 1956), provides an excellent historical example of the use of knowledge about an adversary’s educational background to gain a strategic advantage. Lee received his military training at West Point and was thus familiar with the training received by many of the Union generals (Snow 1987). Lee frequently commented that his tactical decisions were based on knowing the opposing commanders (Davis 1956). Knowledge about cultural tendencies can be used in the chess domain. Due to the intense national pressure and prestige of chess, a Russian chess master will accept the result of a draw in the early rounds of a chess match to escape the humiliation of a loss (Schonberg 1973). This knowledge provides the adversary model the ability to offer a draw with a high probability of acceptance to a Russian chess master. An analysis of 68 games played by Soviet masters and grandmasters during 1973 (Mednis 1978) showed nine specific weaknesses in Soviet play that occurred with statistical significance. Some of the statistical results from the analysis of the Soviet chess games are displayed in table 1. These weaknesses are incorporated into the adversary model as a general knowledge source and are used whenever an adversary of Soviet nationality is encountered. The inclusion of knowledge sources on the cultural and educational strengths and weaknesses for classes of adversaries into the adversary model greatly increases the effective range of situations in which the model can accurately predict an adversary’s probable action.

The adversary model now has knowledge that is specific to a particular adversary’s tactical and strategic decisions and knowledge about the cultural and educational background that affects the tactical and strategic decisions of general classes of adversaries. A third knowledge source provides information about the tactics and strategies that are common to all adversaries in the domain. This third knowledge source is analogous to the opening book moves and evaluation functions used by most chess programs. General knowledge about domain strategies and tactics is used by the adversary model to suggest counter-strategies, such as how to develop the board into an open position when the adversary has demonstrated a preference for closed positions. This domain general knowledge permits the adversary model to make predictions about an adversary’s actions in situations that are novel to the adversary and not influenced by the adversary’s cultural or educational background. Information from the knowledge sources is ordered within the adversary model so that the most specific information available about an adversary that pertains to the current board situation is used to predict the action of the adversary.
CONCLUSIONS

The three knowledge sources discussed previously provide a reliable and robust model of an adversary’s tactical and strategic decision-making process. When implemented into tactical and strategic planning programs, the adversary model will improve the performance of these programs. The strength of the model increases proportionately with the amount of relevant knowledge contained in the model. Additional knowledge sources are still available to the adversary model. One such knowledge source would provide the current age and health of an adversary and the effects of age and health on an adversary’s practical performance. As new knowledge sources are added to the adversary model, the model will continue to increase its capabilities.

<table>
<thead>
<tr>
<th>STRATEGIC KNOWLEDGE</th>
<th>GAMES LOST FROM AFFECT OF STRATEGIC KNOWLEDGE</th>
<th>PERCENTAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lose more with Black pieces.</td>
<td>39 of 68</td>
<td>57%</td>
</tr>
<tr>
<td>Weak middle-game players.</td>
<td>44 of 68</td>
<td>65%</td>
</tr>
<tr>
<td>More errors made from defending than attacking positions.</td>
<td>52 of 68</td>
<td>76%</td>
</tr>
<tr>
<td>Suffer from time pressure.</td>
<td>15 of 68</td>
<td>26%</td>
</tr>
<tr>
<td>More errors due to strategy than due to tactical play.</td>
<td>41 of 68</td>
<td>62%</td>
</tr>
</tbody>
</table>

Table 1: Weaknesses of specific cultural (Soviet) chess education.

REFERENCES


ABSTRACT

One of the most significant problems currently facing the United States is the loss of scientific expertise. Many of the pioneers in such technical fields as flight medicine, ground operations, space propulsion, and safety began their careers during or shortly after WWII and are now approaching retirement age (Keller 1988). While expert systems are recognized as good vehicles for capturing technical knowledge, expert system developers often have difficulty acquiring the knowledge needed for the systems. Because experts in most domains are rare and their time is limited, knowledge acquisition is a significant obstacle to system development. In many cases, however, the baseline knowledge for a system has already been captured in technical manuals written by the experts themselves.

This paper describes an innovative prototype system developed for the Safety and Environmental Protection Subcommittee of JANNAF, the Joint Army-Navy-NASA-Air Force Propulsion Committee. The system, known as the Explosive Hazards Classification Expert System, is unique because of the knowledge acquisition strategy used during its development and because of its integration of expert system and hypertext techniques.

1. AN INTRODUCTION TO EXPLOSIVE HAZARD CLASSIFICATION

The Department of Defense (DoD) and the Department of Transportation (DOT) require explosive materials be assigned labels which categorize their characteristics and hazards (Figure 1). The term “explosive hazard” is a broad term comprising a variety of items which could harm people or damage property, e.g., high explosives, detonators, ammunition, propellants, fireworks, and incendiary devices. The DOT and DoD labels dictate how materials should be stored and transported to minimize danger.

Detailed procedures have been developed for determining the DoD and DOT classifications for explosive materials. The DoD and DOT procedures and tests are described in the Department of Defense Explosives Hazards Classification Procedures (Headquarters, Departments Of The Army, et al. 1981). This document is commonly referred to as Technical Bulletin 700-2 or TB 700-2, the Army version of an inter-service document.

In its present form, TB 700-2 contains fundamental information about the testing and classification of explosives. It provides a good introduction to the process of explosive hazards classification and it is a good reference tool. However, TB 700-2 does not capture the nuances of hazard classification. For example, though each of the services uses a document similar to TB 700-2, they interpret it differently with some more conservative than others. In addition, the experts who routinely classify substances know much more than is contained in the manual. Because substance testing is expensive, the experts routinely classify new materials by comparing them with similar items whose categories have already been determined. This is termed “classification by analogy.” TB 700-2 does not address classification by analogy.

---

Figure 1: Basic Procedure for Classifying Explosive Materials
2. OBJECTIVES FOR THE EXPLOSIVE HAZARDS CLASSIFICATION EXPERT SYSTEM

Two objectives influenced development of the Explosive Hazards Classification Expert System. One objective for the system was to assess the feasibility of a different knowledge acquisition strategy. Traditional wisdom holds that knowledge acquisition is the bottleneck for implementing knowledge based systems. The standard approaches for building expert systems are costly and time-consuming since knowledge engineers and domain expert(s) must spend considerable time working together, particularly at the front end of the development cycle. Our approach was to use TB 700-2 as the basis for early prototypes of the system and to use the experts only as needed to focus development.

A second purpose was to fully integrate an expert system with hypertext (Smith and Weiss 1988). We envisioned a system that would provide hypertext access to both TB 700-2 and supplemental information and would apply expert knowledge to classify hazardous materials. Our view of integration goes beyond simply having a hypertext module and an expert system in the same application. The concept for a three part system (the manual, supplemental expert knowledge, and the expert system) has implications for many aspects of system development ranging from knowledge acquisition to rule base development and the implementation of explanation capabilities.

Figure 2: Rapid Prototyping for the Explosive Hazards Classification Expert System
3. AN OVERVIEW OF SYSTEM DEVELOPMENT

The standard methodology for developing a knowledge based system is rapid prototyping. Rapid prototyping is an iterative process with three steps: knowledge acquisition, prototype development, and review of system performance. We adapted rapid prototyping to suit the challenges of developing a system without extensive access to the experts (Figure 2). A major difference in our approach is that we de-emphasized primary knowledge acquisition from experts for prototype development. We used TB 700-2, the guidance of a Technical Steering Group, and a minimal number of interviews by telephone with a small number of experts.

The implementation effort had two prototyping cycles. The objective for the first prototype (a one month effort) was to establish a sound system concept. The starting point for this version was TB 700-2 and discussions with the customer’s Project Manager about system functionality. Our approach for this phase focused on developing a working version of the human interface. Since our purpose was to demonstrate concepts (not to write code) we used Hypercard on a Macintosh for this prototype even though the target machine for the system was an IBM PC. The prototype demonstrated all of the concepts to be incorporated in the system: it included sections of TB 700-2 in hypertext, examples of supplemental knowledge from experts, and active buttons demonstrating expert system initiation.

Developing the first prototype taught us several lessons. First, as is often claimed by advocates of rapid prototyping, there is real value in building a prototype early in a project, even if this version is thrown away later. The first prototype is important to customers because it provides a more concrete example of the system than just words and pictures. It is important to developers because it forces them to confront issues related to the functionality and design of the system.

The second prototyping cycle (a 5 month effort) had a different emphasis than the first. Following system concept design, we initiated development of a fully functional version for the target hardware: a PC/AT class machine. This implementation included a user interface, an expert knowledge base and a hypertext module incorporating the manual and supplemental knowledge from the experts. The knowledge base was implemented in NEXPERT OBJECT and the hypertext tool used was GUIDE2 developed by Owl International.

The challenging part of this phase was knowledge acquisition. The basic framework for the rule base came from TB 700-2, but we noted omissions and apparent inconsistencies in the manual. With guidance from the Technical Steering Group, we identified several experts who agreed to help develop the rule base and supplement the hypertext module. Given constraints on time, funding, and the need to limit knowledge acquisition, the only feasible means for resolving questions about hazards classification was by telephone and facsimile.

This knowledge acquisition methodology was effective for implementing this system for two reasons. First, TB 700-2 served as a primer for the developers; much of the preliminary knowledge acquisition required for many systems was not necessary for this problem since the manual covered the basics. Secondly, the Technical Steering Group insured that the system concept was sound, and helped us gain the cooperation we needed from the experts. At least for early prototypes, this approach is useful for controlling costs and minimizing the impact of system development on the experts.

Most certainly there are corresponding disadvantages. One problem is that a system developed this way does not "belong" to any single or small group of experts. Our experience is that any newly developed system needs an advocate - a "champion" - if it is to be accepted well by a broad community of users.

A second problem is knowledge acquisition at a distance makes communication between the experts and knowledge engineers (an awkward process under the best circumstances) even more difficult. Structuring conversations with the experts in advance is extremely important. For example, the knowledge engineer and the expert should be familiar with the specific information to be discussed during the telephone interview prior to the conversation. We were most successful when we gave the expert a written copy of specific questions to be discussed during the interview. Furthermore, we requested the expert to provide written responses to the queries in addition to discussing the questions and answers during a telephone conversation.

This approach has several advantages. It minimizes the likelihood of lengthy unproductive telephone conversations since both the knowledge engineer and the expert know what is to be accomplished during the interview. It also minimizes the time required of the expert since the interview is focused on specific questions. Furthermore, the answers the experts provide are likely to be more complete.

Requesting both written and oral responses to the questions is also beneficial. Although the expert is likely to be more verbose during the telephone interview providing the knowledge engineer with more information, the process of writing down the answers forces the expert to evaluate his responses. Written responses tend to be more accurate, and knowledge engineers are less likely to misinterpret a written statement. Many of the experts’ written responses were included as supplemental knowledge in hypertext.

4. KEY ISSUES RELATED TO THE INTEGRATION OF EXPERT SYSTEMS AND HYPERTEXT

The system concept for the Explosives Hazards Classification System integrates multiple kinds of knowledge so that a user can immediately access any available information. The system represents knowledge in three forms: (1) a hypertext version of TB 700-2 augmented with (2) supplemental knowledge from the experts and (3) an expert system module for classifying hazards. The key word behind this concept is "integration". What does it mean to build a system that "integrates" hypertext and an expert system, both from the user’s point of view and from the perspective of system implementation?

4.1 A user’s perspective

From a user’s point of view, the prerequisite for an integrated system is a human interface which ties together all parts of the system. The Explosive Hazards Classification System uses a window-based, mouse driven interface with three basic functions. Users can reference TB 700-2, enter data related to tests performed on hazardous materials, and invoke the expert system to classify a substance.

The principles behind this interface are an extension of the ideas which are commonplace today in interfaces for expert systems. The idea is the interface should "seamlessly integrate" the system’s components: users should be able to
navigate from one function to another with minimal effort. Making the concept work requires anticipating the paths users will need to "navigate" through the system. The Explosives Hazards Classification system supports four pathways among modules. These four pathways are (1) through the system menu bar, (2) through mouse sensitive buttons in hypertext, (3) through data entry screens, and (4) from the expert system to hypertext.

One major path between modules is via the system menu bar (Figure 3). For example, two routes to hypertext exist: a Table of Contents menu for moving to specific chapters in TB 700-2 and a "Views" display for moving to sections of the manual which address specific functions of interest to an analyst. The system menu bar is a stable reference point for users; it is a way of making sure that users are never completely lost in the system.

![Figure 3: System Menu Options](image)

A second level of navigation is through mouse sensitive buttons within hypertext (Figure 4). Reference buttons in hypertext (indicated by bold italic words in the text) point to text or figures in TB 700-2 which are logically related to the bold text. Reference buttons are a basic means for moving from place to place in the document. Note cards (indicated by bold print in the text) are the means for capturing supplemental expert knowledge. To make sure there is no confusion between the "official" TB 700-2 and supplemental knowledge, each note card is annotated with the name and affiliation of the expert who provided the information.

![Figure 4: A Representative Hypertext Display](image)

Data entry displays provide a third means for navigation. Some buttons in the data entry screens link to hypertext entries which provide information about particular aspects of hazards classification. These links allow users to resolve questions about the data being entered. For example, a user might need to know the difference between "explosion" and "detonation"; two terms which describe the behavior of a substance during many hazards tests. Other buttons link to the hazards classification expert system so that the user can generate a hazard classification immediately following data entry.

### CHAPTER 5

**REQUIRED TESTS**

5-1. General.
   a. This chapter 49 Code of Federal by STANAG No. 4117.
   b. Lethal and inert testing under criteria.
   c. "Binary chemical munitions" will be classified in accordance with their industrial chemical hazards until the essential components are combined with their explosive components at which time paragraph b above applies.

5-2. Transport Tests.
   a. These tests measure the commerce of any explosive liquid explosives and stability and sensitivity.
   b. Laboratory samples are used, with weights and dimensions limited to the minimum necessary for drawing conclusions.

Links from the expert system back to hypertext, primarily for explaining the results of knowledge based reasoning, represent a fourth path between subsystems. Our approach combines two types of information, each of which is hierarchically organized (Figure 5). The first type of explanation is a concise textual description of the results generated by the expert system. Another form of explanation is a pointer to hypertext, specifically sections in TB 700-2 (or supplemental expert knowledge) which relate to the conclusion made by the expert system. The system records these explanations as it executes rules, and the user can trace backwards through the decision tree to get increasingly specific information about a substance classification.

4.2 An implementation perspective

We have described a system concept which treats multiple types of knowledge as resources that are equally available to a user through an integrated interface. There are two broad implications related to this approach for knowledge based systems. The first issue concerns development tools for hypertext / expert system applications, and the second deals with the integration of knowledge in a hybrid system.

4.2.1 Development tools. If the objective for a system is to integrate hypertext and an expert system, then a goal for the earliest stages of development is to create a tool for linking different kinds of knowledge -- a knowledge acquisition tool specifically designed for supporting development of a hybrid system. There is a useful analogy between the kind of tool required during the early stages of system development and the need architects have for a sketch pad during the early design of a house. Paper and pencil are useful to an architect because they are flexible and forgiving; early in development of a hypertext / expert system, knowledge engineers need the same kind of tool.

For implementation of a relatively small application like the Explosives Hazards Classification Expert System, we advocate development of a simple, low-cost tool in hypertext,
for example Hypercard on a Macintosh. Figure 6 illustrates the concept. The display has five major elements:

* a Notes area for capturing information from face-to-face or telephone interviews with experts.

* Reference fields for identifying the engineer(s) involved in knowledge acquisition and information about the expert (name, address, telephone number, date of interview).

* a Categories field for classifying the information in the entry. For example, a note in the explosive hazards domain might belong to the class "Thermal Stability Tests".

* a Pseudocode Rule section for representing the contents of a note in a form that can be easily translated into the syntax of expert system rules, and

* pointers to a hypertext manual that provide information related to the knowledge documented in the note.

The intent of this tool is to facilitate capturing, organizing, and linking knowledge very early in development. Over the long term, the tool can be effective for documenting in detail the knowledge on which the system is based. For example, it might be quite useful for an engineer responsible for system maintenance to be able to refer back to original notes made during knowledge acquisition, particularly since these notes may have been expressed as rules in the expert system. The backward link from a rule to its source can be catalogued in comments attached to rules in the knowledge base.
4.2.2 Knowledge integration. Our hybrid expert system presents new challenges in system design, testing and validation. An application like the Explosive Hazards Classification Expert System poses interesting problems because it integrates multiple knowledge representations. Some knowledge in the system has been "officially blessed" (i.e., anything in TB 700-2) while other knowledge represents the viewpoint of an expert. Knowledge from the experts can assume three forms:

- it may represent a clarification or expansion of information from TB 700-2,
- it may reflect an individual expert's opinion, or
- it may illustrate the bias of one of the services responsible for classifying explosives.

In hypertext we deal with this problem by explicitly labeling all supplemental knowledge so that users can make an informed evaluation of the information. Including supplemental knowledge in the system's rule base poses bigger problems. Should the rule base reflect an expert's opinion or the bias of a branch of the service? In the case of explosive hazards classification, a "generic" system (one that does not reflect opinions or bias) would probably not be useful because its functionality would be limited. By the same token, users need a way to control the inferencing process—an Army analyst does not want an Air Force bias to affect hazards classification. For an operational Explosive Hazards Classification Expert System, we envision a segmented rule base that distinguishes between heuristics applicable to one branch of the service and those for another. Part of the user's responsibility is to choose an inferencing strategy: should the data be processed using only baseline TB 700-2 knowledge or using the supplemental rules commonly employed by experts in one (or more) branches of the service? This approach complicates development and puts more responsibility on the user, but it produces a much more powerful and useful system.

5. CONCLUSIONS AND RECOMMENDATIONS

The goals for this development effort were to evaluate the feasibility of an alternative knowledge acquisition strategy and to explore techniques for integrating hypertext and expert systems. A specific objective for the project was to reduce the "knowledge acquisition bottleneck", particularly the need for

---

**Figure 6: Concept for a Simple Knowledge Acquisition Tool**

- **Date:** December 19, 1989
- **Notes:**

  There is a difference between "detonation" and "deflagration". Explosion is often used only in reference to a "detonation". Note that the government defines an explosion in this context as either a detonation or a deflagration.

- **Pseudocode Rule**

  IF the test result is either a detonation OR a deflagration
  THEN an explosion has occurred.

- **Categories:**
  - DoD 1.1
  - DoD 1.2
  - DoD 1.3
  - DoD 1.4
  - DOT Class A
  - DOT Class B
  - DOT Restricted
  - DOT Forbidden
  - Storage Tests
  - Storage Comp. Criteria
  - Transport Tests

- **Expert:** John Smith
  **Agency:**
  **Telephone:** 800-555-5000

- **Document:**
  - Section 5-2a of TB 700-2

- **Knowledge Engineer:** RSS

---
numeorous, time-consuming interviews with experts. With
guidance from a Technical Steering Group, we were able to
implement a useful prototype system based on TB 700-2. To
clarify ambiguities and omissions in the document, we
gathered knowledge from a few selected experts by telephone.
Although development of a fully operational system may still
require extensive knowledge acquisition, the methods we used
are an effective alternative to standard knowledge acquisition.

A second objective for the effort was to fully integrate
an expert system and hypertext. The Explosive Hazards
Classification Expert System links knowledge represented
three different ways: a hypertext manual, supplemental expert
knowledge (also in hypertext), and an expert system. Even if
the expert system module of a prototype is only partially
functional, capturing a document in hypertext has immediate
benefits for a user. Enhancing a document with supplemental
knowledge gives users access to information that normally
would not be available. This integrated approach enables
development of powerful systems that let users perform a
broad range of tasks on one machine.

Based upon our experience, we have five major
recommendations for developers of expert systems. First, we
encourage expert system developers to make extensive use of
existing manuals and technical publications that provide
information about the subject problem. These documents are
quite valuable in the early stages of development since they can
be used by the knowledge engineers as primers and can be
used as the basis for early prototypes to demonstrate system
concepts. These documents also can be used to develop the
initial knowledge base for the system. The knowledge
engineer can then use this knowledge base as an effective
means of developing specific questions to acquire additional
expert information. This has the benefit of minimizing the
time required of the domain experts in the early stages of
system development.

We also recommend developers create knowledge
acquisition tools to facilitate capturing, organizing, and linking
knowledge. To maximize utility, the tools should be easy-to-
use and flexible. Otherwise, the tool will become an
annoyance, not an asset. A simple, low-cost tool in hypertext
should be sufficient for many applications.

We encourage developers to make use of telephone
interviews to reduce knowledge acquisitions costs. If the
knowledge engineer and the domain expert are both familiar
with the specific information to be discussed during the
telephone interview prior to the conversation, the interview can
be very productive, reduce travel requirements, and minimize
the time required of the expert.

We also recommend the integration of expert system
techniques and hypertext. This synergistic approach has two
significant advantages for the system user. Using the
hypertext capabilities of the system, the user has access to
information that is important to the problem at hand including
supplemental knowledge not suitable for expert system
reasoning. In addition, using the explanation function, the
user can review the system's decision making process as a
function of the associated technical manual.

We also encourage developers to be sensitive to the
issue of the time required of the end users to review and
critique the system. Although developers typically schedule
their own time and resources for system reviews, it is equally
important for the end users to do the same. Their comments
and suggestions about the system during system development
are paramount to project success. It is the developers
responsibility to ensure that the end users understand their role
in the development process and allocate their time and
resources appropriately. If this aspect of system development
is managed appropriately, it will not only facilitate system
development, but will also facilitate system acceptance.

6. REFERENCES

Headquarters, Departments Of The Army, The Navy, The Air
Of Defense Explosives Hazard Classification Procedures.

Learned": A Systems Approach." Proceedings, JANNAF
Safety and Environmental Protection Subcommittee Meeting.
Chemical Propulsion Information Agency, Laurel, Maryland,
453-459.

Hypertext." Communications of the ACM 31, no. 7 (July):
816-819.

48
HUMAN ENGINEERING THE
KNOWLEDGE ACQUISITION INTERFACE:
THE EVOLUTION OF VISTA

Robert Ahlers
Naval Training Systems Center
Human Factors Division
Orlando, FL 32826

Mark Schnitzius
Oak Ridge Associated Universities
Oak Ridge, TN 37831

ABSTRACT

An important aspect in the design of knowledge acquisition tools is the user/computer interface. This aspect has generally been ignored, despite the fact that the process of knowledge transfer places unusual demands upon the user. This paper describes the design considerations from a user's perspective and the manner in which the design was implemented in an evolving knowledge acquisition tool, the Visual Interactive System for Task Analysis (VISTA).

BACKGROUND

The goal of knowledge engineering is to produce an executable representation of a self contained "chunk" of knowledge within an application domain. This computer resident representation is the end result of transforming the knowledge content through at least two other representations. Initially, the knowledge resides in a cognitive structure within the head of a domain expert. The second representation contains the knowledge when it is in an external form, being transmitted from the domain expert to the knowledge engineer. This is an intermediate form and is generally visual. For example, it may be in the form of handwritten notes or in source code on a display screen.

Knowledge engineering involves activities associated with transforming knowledge among the various representations. The acquisition of domain knowledge for the purpose of knowledge engineering is an ill-defined and inefficient process (Hoffman, 1987). A computer-based tool to aid knowledge acquisition could potentially alleviate both shortcomings. However, there are many unsolved issues associated with such an aid (Bareiss, Porter, & Murray, 1989). Some issues address database aspects of knowledge engineering such as the computational efficiency of various knowledge representations. Other issues concern the transformation of the knowledge content between representations. One of these transformations takes place between a human's cognitive representation and the initial computer-based representation. The efficiency with which this transformation takes place is largely dependent upon the user/computer interface (UCI). The design of the UCI is the major focus of this paper.

An efficient UCI is particularly critical for a knowledge engineering tool because a large amount of knowledge is typically represented in an expert system, and it all must be entered through the UCI. The interface must allow the user to efficiently enter, browse, and modify the domain knowledge. "Efficiently," as used here, refers to minimum keystrokes, easy access to views of the evolving knowledge in diverse formats, and maximum contextual cues. The design of such an interface is a particularly challenging task.

One approach to the design and development of a UCI which meets these objectives is described in this paper. The interface, implemented in a software tool for analysing expert job performance is known as the Visual Interactive System for Task Analysis (VISTA).

DESIGN PHILOSOPHY

VISTA was designed in accordance with two basic principles of human information processing. First, people receive the majority of information through the visual sense. Pictures efficiently summarize a great deal of information. They provide a static display of information which allows the viewer to shift the focus of attention to currently important aspects. The second principle is that people have limited information...
processing capabilities. To deal with this limitation, people naturally group or "chunk" related pieces of information into hierarchical structures of varying levels of abstraction. The VISTA interface capitalizes on these two principles.

The following design goals were set in accordance with the design philosophy:

- Require minimum keyboard input
- Allow incremental refinement of the knowledge base
- Provide multiple views of knowledge through both graphic and text displays
- Allow separate entry of procedural and declarative knowledge
- Incorporate operating cues in the interface
- Maximize the use of contextual information

These goals were met by using a windows-icons-mouse-pointing interface to encourage rapid selection and execution of system functions, minimize user keystrokes and provide a design with obvious icon and menu selectable functions. Rapid input of procedural information is further facilitated by the capability to import knowledge created by other software. A graphical network provides a structure for representing procedural knowledge and also a mechanism for attaching declarative information to individual task elements. Both sequential and concurrent relationships among procedural tasks can be shown. Decision tasks and repetitive procedures can be represented. A graphic display of the full knowledge base hierarchy provides context. The interface design encourages the user to iteratively construct the knowledge representation, proceeding in either a top-down or bottom-up fashion.

There is a great deal of literature on UCI (e.g., Schneiderman, 1987), but surprisingly little information is sufficiently detailed to guide engineering design (Norcio & Stanley, 1988). For this reason, the development process of VISTA included periodic formal evaluations by potential users. The evaluations documented user performance on the prototypes and solicited comments and suggestions for improvements. VISTA was programmed in Smalltalk, a language which facilitated rapid changes in the interface design.

FEATURES

VISTA's capabilities are accessed with mouse-sensitive icons and menus. Figure 1 illustrates the layout of the primary VISTA screen with an example of a top level knowledge base. At the top is a message area that identifies the current domain, the current level in the hierarchy which is being shown, and the domain number.

![Figure 1. Primary interface screen, with example task domain loaded.](image)
Icons, representing frequently used functions, are presented in a column along the left-hand side of the screen. The eight system functions which are accessed by "clicking" with the mouse on icons are (from top to bottom on Figure 1):

- Create a task within a procedure
- Indicate a temporal sequence between tasks
- Move one or a group of tasks from one location to another
- Delete one or a group of objects
- Copy one or a group of tasks from one location to another
- Scroll within a level
- Identify current position in the hierarchy and accommodate movement among levels
- Solicit help

Less frequently used functions reside on nested pull-down menus accessed by clicking on one of the four menu titles (JOBS, EDIT, SPECIAL, or OUTPUT) near the top of the screen. Groups of functions are provided for inputting, editing, browsing, and outputting information in various formats. Along the bottom of the screen is a message area which displays statements for aiding the user.

OPERATION

To construct a procedural network, a user positions a set of tasks on the screen and then links them with arcs to indicate the sequence of activity. A user gives each task a short description by either entering it from the keyboard or by selecting words or phrases from a series of menus using the mouse. Any task may be described in more detail by clicking on it to allow the construction of a lower level procedural net. There is no practical limit to the depth of task detail which may be represented. Tasks with lower level nets are differentiated with color coding from those without lower levels. In Figure 1, the two darker task icons contain lower level nets. Placing the cursor over one of these tasks and clicking the middle mouse button causes a display of its next lower level. Clicking the middle button on empty space causes the next higher level to be displayed.

The provision for structuring complex procedures into subnets is a major advantage of the VISTA UCI design. This feature allows a user to create and view domain information at various levels of abstraction. This helps a user to organize the knowledge to match his cognitive structure. The knowledge "tree" may be easily browsed by clicking the middle mouse button as described above, or by calling for an overview of the entire structure (Figure 2) and clicking the left button on the desired node.

In addition to the procedural information contained in the network representation, a user can store declarative information related to task performance. This

![Figure 2. Overall view of knowledge structure, initiated by clicking on "network" icon near bottom of icon pane.](image-url)
Figure 3: Declarative knowledge entry screen showing an example of a numerical scale entry.

Information may be accessed, for entry or inspection, by pointing to a task and clicking the right mouse button. This causes a list of declarative information categories to appear in a menu. One of these may be selected, as seen in Figure 3 or a new category may be easily added by the user. Each category may be associated with a predefined data type. Currently implemented types are text, Boolean, pre-enumerated list, and numerical scale.

**APPLICATION**

The major thrust of the project has been to create an environment which allows an expert to represent knowledge as naturally as possible. VISTA's development cycle has included three formal evaluations using domain experts. It has received high scores on user rating scales and also enthusiastic user comments. VISTA has been highly successful in providing an efficient mechanism for depicting and maintaining procedural and declarative knowledge that defines an application domain.

A knowledge acquisition tool must minimize the disparity between an expert's internal knowledge representation and that imposed by the tool (Gruber & Cohen, 1987). That a sample of domain experts has found VISTA easy to use suggests that its disparity is acceptable.

**SUMMARY**

The research objective of providing an efficient GUI for knowledge transfer has been met. The VISTA design provides a set of constraints which will guide further development toward a more complete knowledge acquisition tool. The next phase of development will focus on providing the functionality to transform the surface, graphic representation into an executable knowledge base.

**REFERENCES**


A COMPARATIVE ANALYSIS OF THREE TECHNIQUES OF KNOWLEDGE ACQUISITION FOR EXPERT SYSTEMS APPLIED TO THE DOMAIN OF LITERARY PREFERENCE

Je-Yien Ernst
Eckerd College
St. Petersburg, Florida

ABSTRACT

This paper provides a comparative analysis of three knowledge elicitation techniques applied to a novel domain, that of literary preference. By comparing the advantages and disadvantages of certain techniques, I hope to assist the expert system developer in choosing a suitable acquisition method. Techniques applied to my literary preferences include the manual approach, implemented with Quinlan's ID3 algorithm (Thompson and Thompson 1986), a more automated approach using an SPSSX (Statistical Package for the Social Sciences) program to determine relevant attributes, and the fully automated method of machine induction from a table of values.

I. INTRODUCTION

Expert systems are knowledge-based intelligent programs containing the interpreted and organized information which experts rely upon to achieve such high degrees of proficiency. These systems are of extreme importance in training novices, preserving expertise for future generations and providing expert assistance when a human expert is unavailable. Eliciting knowledge from an expert is perhaps the most time consuming and economically costly part of the development of an expert system, and hence the relative accuracy and efficiency of different techniques, as well as the relative merit of the rules they may produce, are a matter of concern and are the subject of this paper.

Among the costly operations that contribute to knowledge acquisition problems are those of consulting the expert, finding an appropriate representation within which to express his/her expertise, and finally, converting the raw elicited data into that representation (McGraw and Harbison-Biggs 1989). These problems require the knowledge engineer to be quite skilled in various areas other than that which he or she may be specifically trained for, i.e. the development of a computer program. Automating the process of knowledge acquisition can eliminate these problems and result in a more effective knowledge base. This paper will provide a comparative analysis of several knowledge elicitation techniques applied to a novel domain, that of literary preference.

II. MANUAL APPROACH

First, I sought to identify the set of attributes relevant to my "output" reaction to a work of literature that could be quantified in a reasonable way. I then evaluated each of a fairly eclectic selection of literary works with respect to such input attributes as: perceived scholarliness, length, ability to keep attention, ability to create suspense, comprehensibility, predictability, informational value, reader's purpose, redundancy and a final rating.

Quinlan's ID3 algorithm was used to create classification trees from which IF/THEN rules were developed (Thompson and Thompson 1986). This represents a manual induction method whereby knowledge is obtained directly from the data (Thompson and Thompson 1986). To create a classification tree, one variable is chosen, in this case an attribute listed above, and all the examples are grouped according to their rating for that particular attribute. Shown below is part of a tree classified according to the suspenseful category.
If each group has the same final rating (the first number in each line), then that branch is finished. If not, then pick another attribute by which the examples can be further broken down, as indicated below.

If suspenseful = 1

- easy reading = 1
  THEN final rating = 1

If suspenseful = 2

- easy reading = 2
  THEN final rating = 4.

This process is continued until each partition contains examples with the same final rating. Rules are then created by following each branch until it ends. For instance:

IF suspenseful = 1 AND easy reading = 2
THEN final rating = 1

A classification tree was created for each attribute, resulting in a knowledge base containing a total of 142 rules. In most instances, the rules generated compared favorably with my own opinion of the books. For example, if the book has a high scholarly rating, keeps my attention and is highly suspenseful, then the final rating will be high, 5 on a scale of 1 to 5. On the other hand, if the book has a low scholarly rating, with a predictability rating of 3 (5 being the highest), and is not extremely suspenseful, then the final rating will only be 3, not as high as in the previous example.

Sometimes, however, the results from the tree were not as I anticipated. In the suspenseful category of 3 (4 being the highest), a book with a lower easy reading value had a higher final rating, while a book with a higher easy reading value had a lower final rating. A different final rating than what I expected occurred between one and three times in each classification tree. This suggests a number of possibilities.

Most likely, there are some other attributes, in addition to the ones which I have enumerated, that affect my final rating. Subject matter is one aspect that could influence the final rating regardless of other attribute values.

The length of time since I read a particular book could affect the results in each category as well. This could very well lead to inconsistent results in the trees. Also, I have no standard, as far as I can tell, for determining attribute values, which meant that it was sometimes hard to assess the category. These standards represent a deeper form of heuristics, or rules of thumb, for each category, and I was unable to determine them with any of the three acquisition techniques.

Additionally, the size of my data could have an effect on the type of rules generated. A few rules were formed which do not particularly match my beliefs, and if more examples were available, different rules might be generated which more closely represent my own opinion. For example, there was only one book with a suspenseful rating of 2, so a rule was created as follows:

IF suspenseful = 2
THEN final rating = 4.

If more examples were in this category, several rules, leading to different final ratings might result.

Finally, as the trees were created, it was hard to determine the most relevant categories by which to partition the examples. Totally different knowledge bases can be created by partitioning examples based on different attributes.

III. MANUAL + AUTOMATED

To alleviate some of these problems, I combined the manual technique with a more automated approach involving a statistical analysis of the input attributes. I wrote a program in SPSSX to determine which were the most important factor(s) about a book according to data supplied by the "expert" (myself). Correlation coefficients for all the variables were computed in order to ascertain any important relationships between the attributes and the final rating. The novel's ability to keep my attention has the strongest correlation with the final rating, while the book's length, surprisingly, has the second highest value. This result would be an
interesting issue for further study because it is not evident to me that the length of a book affects how much I enjoy it, yet length turned out to be statistically significant. Other important variables, according to the SPSSX results, are easy reading, suspenseful and easy to get into, in that order.

These results were used to delete the irrelevant variables and rules from the knowledge base created by the manual approach involving the ID3 algorithm. Intuitively, it would seem that an expert system using this modified knowledge base would come up with more accurate results if insignificant attributes (according to the SPSSX program) were not included. When the expert system is calculating the final rating, the first rule that matches the input is used to determine the result, so if irrelevant variables are placed before others, the result could very easily be incorrect. Therefore, insignificant attributes were deleted from rule conditions, and those beginning with such a variable (IF informational value = 3 AND...) were deleted altogether. The resulting knowledge base contains 62 rules, slightly less than half the number created with the classification trees.

IV. AUTOMATED APPROACH

A third knowledge base was created automatically using a rule-based expert system development tool. The rating grid was entered into the expert system shell, creating an induction table similar to this:

<table>
<thead>
<tr>
<th>School</th>
<th>Len</th>
<th>Sch</th>
<th>Purp</th>
<th>Easy</th>
<th>Susp</th>
<th>Pred</th>
<th>Redund</th>
<th>Final</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

The tool automatically creates a knowledge base from the table of examples. One rule is created for each line in the table. For instance:

IF scholarly = 2 AND length = 3 AND attention = 3 AND get into = 3 AND purpose = 3 AND inform = 3 AND easy reading = 5 AND predict = 1 AND redund = 1 AND suspend = 1

THEN final = 5

A knowledge base containing 24 rules was created in this manner.

V. RESULTS and CONCLUSIONS

I entered the two knowledge bases, which I created myself, into the same expert system shell used to create a third knowledge base. In effect, three expert systems were developed with which I could test my new examples and determine which system produced more accurate results.

Testing proved to be slightly difficult as I was only able to provide five new examples of books. In order to increase the number of testing examples, I discussed the books with a classmate who has read many of the same books. We discussed my attribute ratings and found that, in almost every instance, they were the same ratings as she would have given. Our rating systems are comparable enough that the three knowledge bases can be sufficiently and accurately tested for the purposes of this project.

According to the results, the manually created knowledge base provided the most accurate ratings, followed by the SPSSX-influenced system and then the automatically induced knowledge base. This last one was unable to determine any final ratings. After seeing how the development tool automatically forms rules from the table, I did not expect this approach to produce any results. For a final rating to be reported, every category of the new example had to match exactly with an example already present in the knowledge base. It does not mean, however, that this technique of automatically acquiring knowledge is useless. It is important to remember that not every tool and approach is suitable for every domain. The particular domain I chose does not lend itself to being accurately structured using this particular tool.

The manually developed knowledge base surprisingly resulted in more accurate results than the one obtained from a statistical analysis of the attributes. It was my opinion from the beginning that the latter would be more accurate, and reasons for this unexpected result will be further investigated.

The ID3 algorithm, which I used to create the rules manually, is utilized in a system called MacSmaarts by Cognition Technology (McGrath and Harbison-Briggs 1989). This seems to be an appropriate tool for my domain. Creating a knowledge base using MacSmaarts would certainly be more practical with respect to the time involved with the process. Using a system to automatically acquire knowledge will provide expertise at a fraction of the time and cost required by the manual approach (Hayes-Roth et al 1983). Human resources are steadily becoming more expensive and scarce, so any assistance expert systems can provide is beneficial.
actively elicits knowledge from an expert is called SALT, which can be used by experts in any domain (Marcus and McDermott 1989). Questioning might also be beneficial in ascertaining other relevant attributes since it is evident that something is missing from my rating scheme. Furthermore, a number of books had the same ratings in the categories, yet ended up with a different final rating. A system to query the expert could be advantageous in pinpointing reasons for this. No technique was evident to accomplish this difficult task manually.

VI. SUMMARY

Based on my own comparisons of knowledge elicitation techniques, an automated approach of acquiring knowledge is more efficient and practical than using algorithms manually to create set of rules. It is also feasible, whether the manual or automatic mode is used, to combine techniques and not rely solely on one method in order to result in the most effective knowledge base. Certainly, as Hayes-Roth and others observe (Hayes-Roth et al. 1983), automated knowledge elicitation methods "present a breakthrough in the knowledge acquisition bottleneck familiar in the construction of expert systems" (Hayes-Roth et al 1983).

VII. ACKNOWLEDGEMENTS

Special thanks to Professor Mark Fishman for his help with this project.

REFERENCES


THE COGNITIVE PSYCHOLOGY OF EXPERTISE:  
A REVIEW OF THE RESEARCH LITERATURE

Robert R. Hoffman
Department of Psychology
Adelphi University
Garden City, NY 11530

ABSTRACT
This paper surveys hypotheses which have been offered by experimental psychologists and by developers of expert systems concerning the nature of expert knowledge and reasoning. There is a great deal of relevant research in which many of the hypotheses have been tested. The present paper focuses on the implications of this research for a general, operational definition of expertise, one which addresses both cognitive and functional aspects.

INTRODUCTION
This paper presents a broad overview of the cognitive psychology of expertise. Some of the earliest studies which could be considered relevant to the psychology of expertise were conducted as long ago as the late 1900s. The relevant literature is more extensive than one might suppose. (For detailed reviews, see Burton, Hoffman, Shadbolt, & Klein, 1990; Hoffman, 1990; Hoffman & Deffenbacher, 1990; Hoffman, Burton, Shanteau, & Shadbolt, 1990.) Research pertinent to expertise has been conducted largely within three paradigms in experimental psychology: Judgment and decision-making, problem solving, and learning and memory.

RESEARCH REVIEW
Research on judgment and decision-making has focused on judgment "hit rates" and reasoning biases. Historically, research in this area seemed to show that experts (i.e., physicians, economists, etc.) are no better than anyone else at making predictions or judgments. In this paradigm, it is possible to demonstrate reasoning bias by forcing research participants into unnatural probability estimation tasks. The bulk of the available evidence shows that reasoning biases are not rampant in performance at real-world tasks. Furthermore, the measurement of hit rates ignores most of what's important about experts' knowledge. Thus, the study of biases and hit rates does not form an adequate foundation for a theory of expertise. There are, however, specifiable properties of domains which can make reasoning biases more or less likely to occur.

Expertise has also been the focus of research on problem solving, which uses the "think aloud" protocol analysis paradigm, and research on learning, which uses various memory tasks (i.e., recall recognition). Research has focused on the development of expertise (i.e., the differences between novices and experts) in such domains as computer programming and physics problem solving. This research supports a number of hypotheses about expertise, for example, hypotheses about memory schemas and mental models, and hypotheses about the developmental levels of expertise.

Developmental Levels
The present discussion relies on traditional guild terminology. According to this scheme, a novice is one who is totally ignorant of a domain. The novice (one who is "new") has had some preliminary exposure to the domain. The initiate is one who has begun an introductory course of study. The apprentice is one who has gone beyond the introductory level, is "immersed" in the domain, and assists someone at a higher level. A journeyman is an experienced, competent, and reliable worker (literally, one who can perform a day's labor unsupervised). An expert is a distinguished and brilliant journeyman, highly regarded by peers, whose judgments are uncommonly accurate and reliable, whose performance shows consummate skill, and who can deal effectively with rare or "tough" cases. Also, an expert often has special skills or knowledge derived from extensive experience with sub-domains. A master is an expert whose judgments set the standards, regulations, or ideals.

Traditionally, a master is any expert

Proceedings of the 3rd Florida Artificial Intelligence Research Symposium, 1990, by the Florida AI Research Society

c 1990 FLAIRS 0-9620-1732-9/90/0350-0057

57
It is important to know the developmental level of participants in any given study, since not all the so-called "experts" will be experts, not all the so-called novices will be novices, etc.

HYPOTHESES ABOUT EXPERTISE

To date, consideration of hypotheses about the nature of expertise has been superficial and piecemeal. For example, in their stage theory of the development of expertise, Dreyfus and Dreyfus (1986) confound a number of separable hypotheses (having to do with automaticity and perceptual learning). To date, there has been no attempt to compare and integrate hypotheses across the paradigms within psychology, let alone across psychology and AI. Fortunately, the literature—including ideas presented in case study reports on expert system development projects—can be boiled down to a manageable set of hypotheses about the cognition of experts. These are presented here in Table 1 in name only, since each hypothesis and sub-hypothesis is actually more or less lengthy proposition.

The definition of expertise presented above falls at an operational level. That is, expertise is defined (as much as possible) in observable aspects of behavior or criteria for identifying experts. The hypotheses listed in Table 1 entail refinements of the definition at a cognitive level (What happens in the expert's perception, reasoning, and judgment?) and at a functional level (What purposes are served by the expert's mental operations and skills?).

CONCLUSIONS

Many of the major hypotheses about expertise, such as the Drive Toward Comprehension Hypothesis and the Conceptual Organization Hypothesis, have been amply demonstrated in the laboratory. However, some of the most widely-held hypotheses, such as the Perceptual Learning Hypothesis, have not been adequately investigated. Further psychological research—in problem areas such as knowledge elicitation, representation, explanation, and training—has the potential for contributing significantly to knowledge engineering.

REFERENCES


Table 1. Hypotheses about Expertise

A. DEVELOPMENT HYPOTHESES
1. The Basic Developmental Hypothesis
2. The Developmental Continuum Hypothesis
3. The Developmental Levels Hypothesis
4. The "Drive Toward Comprehension" Hypothesis
5. The Perceptual Learning Hypothesis
6. The Automaticity Hypothesis
7. The Individual Differences Hypothesis
   (a). General Intelligence
   (b). Personality Variables: Communication skill, Self-confidence, Extraversion, Stress management skill.
8. The Contextual Relativity Hypothesis

B. KNOWLEDGE STRUCTURE HYPOTHESES
1. The Conceptual Organization Hypothesis
2. The Knowledge Types and Dimensions Hypothesis
   (a). Content/function Categories: Factual knowledge, Procedural knowledge, Meta-knowledge
   (b). The Generalizability Dimension
   (c). The Accessibility Dimension
3. The Mental Models Hypothesis
4. The Schema Hypothesis

C. REASONING HYPOTHESES
1. The Reasoning Operations Hypothesis
   (a). Deduction and induction, Generalization and specialization, Restatement or reformulation, Metaphor and analogy
2. The Process Sequences Hypothesis
   (a). Top-down and bottom-up reasoning
   (b). The Divide-and-conquer
strategy
   Elimination by aspects,
   Means-end analysis
(c). The Refinement cycle
3. Reasoning Biases Hypotheses
(a). The misuse of information
   Ignoring relevant information,
   considering one
   hypothesis at a time, relying on irrelevant information
(b). The Validity and Reliability Hypothesis
(c). Logical and psychological biases
   Content bias, set inclusion errors, the
   "atmosphere" effect, functional fixedness
(d). The Reasoning Biases Hypothesis
   Confirmation bias, "anchoring," overconfidence, over conservatism,
   the "representativeness" bias, the "wishful thinking" bias, the "cognitive availability" bias.
Towards Computing the Aspect Graph of Constructive Solid Geometry Objects

Randy Bolling
Kevin Bowyer

Department of Computer Science and Engineering
University of South Florida
Tampa, Florida 33620 U.S.A.

bolling@pluto.usf.edu, kwb@sol.usf.edu

Abstract. The aspect graph has recently become a popular topic of research in computer vision. A number of authors have described algorithms for computing the aspect graph for various classes of objects. We describe a first step toward computing the aspect graph of objects defined through a typical constructive solid geometry system (CSG)—the aspect graph of a CSG of two spheres under orthographic projection.

INTRODUCTION

The field of computer vision requires some form of internal representation of 3-D objects. Recently, researchers have turned to aspect graph as a means of this representation. The concept of the aspect graph is generally credited to Koenderink and van Doorn (1979). It is a graph structure in which

- nodes are topologically distinct views of the object within a maximal connected partition of viewpoint space, and
- arcs are visual events that occur in the transition from one region of viewpoint space to another. A visual event is an instantaneous change from one view to another. To understand, take the case of a cube. Only from a limited region of viewpoint space can we see, say, the top face exclusively. Any substantial motion of the viewpoint will cause a second and/or third face to appear.

The algorithms for creating the aspect graph use a geometric model as input. Typical geometric models are boundary surface description, constructive solid geometry and generalized cylinders. When creating an algorithm, one must consider the viewpoint space that encompasses the object. Viewpoint space is usually the Gaussian Sphere for limited cases of orthographic projection, and 3-D space for broader cases of perspective projection. Under Orthographic projection, the aspect graph does not take into consideration the closeness of the object to the viewpoint, and therefore limits the number of views, and the location within which the viewpoint may occur, whereas perspective projection takes such views into account. Another factor considered in creating the algorithm is the exactness of the solution. An exact solution can be found only when the algorithm computes the views based on the geometry of the object. Currently object-based solutions exist for polyhedral objects using both orthographic and perspective projection (Gigué and Malik, 1988; Gigué et al, 1988; Ikeuchi, 1987; Plantinga, 1986; Stewman, 1988, Watts, 1988). Similar solutions exist for solids of revolution using the definition of Right, Circular, Straight, Homogeneous, Generalized Cylinders (RCSHG) (Eggert, 1989; Kriegman and Ponce, 1989). A more complicated class of curved surface objects can be created through the models of CSG objects. Currently there exist numerous CSG modelers (e.g. PADL-2) which use primitive solids such as the torus, sphere, cone and wedge. From these primitives, the basic combing operations of union, difference, and intersection are used to create the final object. It has been stated that 80 percent of all typical unsculpted industrial parts are representable using CSG objects (Samuel et al, 1976). The long-range goal of this research project is to create an algorithm that takes as input a CSG object and computes the final aspect graph. In this paper, we present the solution for a first step towards this goal. We show how to calculate the aspect graph for a limited CSG object using an object-based algorithm and orthographic projection.

PROBLEM DEFINITION

In this first step, we take as our set of objects those that can be created through the interactions of two spherical primitives. To parameterize our spheres, we define them each by their radius and center. In the definition of CSG objects, there are three combining operations: union, difference, and intersection. Consider two spheres of arbitrary size combined in one of these three ways. Their corresponding final objects are shown in Figure 1. We also consider the Gaussian Sphere model of viewpoint space. The North Pole of the Gaussian Sphere is located as shown
in Figure 2. Our goal is to subdivide the Gaussian Sphere into regions within which a view is considered topologically equivalent. In the case of two spheres, we note that any view that exists at a point some angle down from the North Pole, will exist anywhere along the latitude corresponding to that angle. This is due to the symmetry of the object in our case of two spheres. It is important to note that our spheres must have at least some overlapping volume. Furthermore, the combining operation must create an object that has a volume greater than zero.

When two spheres intersect, there will exist a cross-sectional circle \( C_s \) along the curve of their interaction. This is clearly seen in the difference combining operation, where \( C_s \) is the circle along the "hole" of the sphere. With the two spheres labeled 1 and 2 as shown in Figure 3, \( C_s \) will lie either

1. to the Right of Sphere 2's Great Circle, or
2. to the Left of Sphere 2's Great Circle, or
3. exactly along Sphere 2's Great Circle.

Another factor to consider in the interaction of the two spheres is their relative size. They will have one of three possible relationships:

1. Sphere 1 is greater in size than Sphere 2, or
2. Sphere 2 is greater in size than Sphere 1, or
3. Sphere 1 is equal in size to Sphere 2.

From this, we can exhaustively enumerate the 27 possible object types representing different instances of combining operation, \( C_s \)'s relationship to the Great Circle, and the relative size of the two spheres. These 27 possibilities are summarized in Table 1. They actually reduce to just 6 qualitatively distinct cases.

We now discuss in detail the reduction of the 9 instances in Table 1 under the union operation to Union Case A and Union Case B. The other similar reductions are not discussed here due to space limitations. Observing that there are three impossible objects among the nine instances, there are just six remaining instances to explain. From here we see that when Sphere 1 is Sphere 2 and \( C_s \) is at the Great Circle of Sphere 2 (GCS2), a trivial solution of a single sphere is produced. The first significant object is that of Union Case A, which represents four of the remaining instances (\( C_s \) Left of GCS2 will create 3 of three objects, and \( C_s \) at GCS2 will create a limiting case.) The second object, Union Case B, will be created only when Sphere 1 > Sphere 2, and \( C_s \) is to the right of GCS2. The 6 basic objects are shown in Figure 4. The sequence of visual events and the views for the objects are discussed in the next section.

Before describing the views of the objects, we must first define what appears in a view. We assume that the image is composed of two types of contours, limits and edges. A limit results from a contour generator on the object surface. The contour generator is the locus of points on the object's surface where the line of sight is tangent to the surface. An edge in the image results from a crease on the object's surface. A crease is where there is a discontinuity in the surface normal. A final term one must understand is an occlusion. An occlusion occurs when some opaque object (or part of the object) blocks our line of sight from seeing some item such as a crease or contour generator.

**Algorithm**

Noting the symmetry of the object, our final goal is to find angles with respect to the North Pole of the Gaussian Sphere, and from this we will have determined how to properly subdivide the Gaussian Sphere. The first step in this algorithm is to determine which of the 6 basic cases describes our CSG object. Our algorithm then breaks into one of the three combining operations for further analysis.

**Union of Two Spheres**

The union of two spheres allows creation of objects with one of two distinct series of visual events. (To assist in understanding the description of the visual event angles, refer to Figure 6.) We first consider Union Case A. Figure 5a shows a series of views. These are known as stable views since they can be seen over some maximal connected region of the Gaussian Sphere. The points along which there is a transition from one view to the next are the locations of visual events. A visual event is an unstable view. The first view of the object, starting at the North Pole, is a limb created by Sphere 1. This limb occludes all other potentially visible limbs and/or creases created by Sphere 2. As we move towards the South Pole, this view is present until the angle at which Sphere 2 is no longer occluded by Sphere 1. This angle, associated with this visual event, can be calculated by solving for the tangent line common to both spheres. From this line, we take the arc tangent of its slope to determine the angle of the visual event (angle \( \alpha_1 \)). From here, we have a constant view of Sphere 1, with Sphere 2 on the horizon until the point at which the crease formed at \( C_s \) becomes visible. This occurs at the angle related to the line tangent to Sphere 1 and containing a point of \( C_s \) (angle \( \alpha_2 \)). The visual event that occurs is a limb-crease interaction. The net effect is the transition to the next view where we see limbs created by Sphere 1 and Sphere 2, as well as a small edge. The next visual event occurs when the edge reaches its maximal possible length and this always occurs at 90 degrees with respect to the North Pole (angle \( \alpha_3 \)). From here, the edge begins to shrink, giving way to the limb created by Sphere 2 until the next visual event. This visual event is characterized by Sphere 2's occlusion of the entire edge, and its associated angle can be calculated from the line tangent to Sphere 2, containing a point of \( C_s \) (angle \( \alpha_4 \)). After crossing that angle, we see the full limb of Sphere 2, and Sphere 1's limb is partially occluded. We move to a point when the limbs will appear to touch at only one point, and this visual event is located at 180 degrees minus angle \( \alpha_1 \) (angle \( \alpha_5 \)). After this visual event, we maintain the stable view of Sphere 2's limb within Sphere 1's limb. This concludes Union Case
A. Now we consider Union Case B. Referring to the event series diagrams in Figure 5.2, we see the stable views created by this object. The first view of this case contains only the limb of Sphere 1. From here we move to angle $\alpha_1$, where the limb of Sphere 2 becomes visible at a visual event. Then a stable view will be present until $\alpha_2$, when a visual event occurs. The visual event is the presence of the edge at $C_\theta$. This edge increases in length until we reach angle $\alpha_3$, when the edge is at its maximal length. From here a stable view of the edge, beginning to form a complete circle, is visible. The next visual event occurs at angle $\alpha_4$, when the limb of Sphere 2 touches Sphere 1's limb at a single point. After crossing that angle, a stable view is present until a new angle (angle $\alpha_{5a}$). This angle is located at 180 degrees minus $\alpha_4$. At this angle, the edge forms a complete circle along $C_\theta$. From here, the view will be stable for the rest of the event series.

**Difference of Two Spheres**

Noting the event series for Difference Case A in Figure 5.c, we see that the first view is that of the limb of Sphere 1. From here we move to angle $\alpha_2$, where the visual event is the first appearance of the edge. After that, we see a stable view up until ninety degrees, where a visual event occurs at the edge. This event will give way to the next view of a full circle $C_\theta$. From here, we move to a new visual event marked by the tangent of Sphere 1 that touches the edge at $C_\theta$. (This is somewhat similar to angle $\alpha_2$.) This new angle (angle $\alpha_{5a}$) is from a point along $C_\theta$ that is directly opposite the point that $\alpha_2$ passes through, and will thus have a value equal to 180 degrees minus $\alpha_2$. This marks the point where the crease at $C_\theta$ touches the limb of Sphere 1 at one point. From here we move to a completely stable view whose line drawing includes the edge at $C_\theta$ within the limb from Sphere 1.

In Difference Case B (Figure 5.d), we start out with a view of the crease at $C_\theta$. From the North pole a visual event will occur when we reach angle $\alpha_2$, which will cause a limb from Sphere 1 to touch the edge at $C_\theta$. After crossing angle $\alpha_2$, we see a stable view of a partial limb of Sphere 1 occluding part of the crease. From here we move to the next visual event, occurring at ninety degrees. Here, the arc of the edge will become a line. After this angle, we see the return of the entire edge, with a small part of the limb from Sphere 1. From here we move to angle $\alpha_4$, where the edge at $C_\theta$ will first occlude the limb of Sphere 1. From this visual event, or view will remain stable for the rest of the event series, as we are capable of only viewing the edge at $C_\theta$.

**Intersection of Two Spheres**

In Intersection Case A (Figure 5.e), we first see the edge at $C_\theta$, and the limb of Sphere 1 is occluded. From here we move to angle $\alpha_{5a}$, where the limb from Sphere 2 will first become visible. This view then remains stable until angle $\alpha_2$, where the limb of Sphere 1 first becomes visible. The view currently contains both limbs and the edge. The edge at $C_\theta$ will shrink until it reaches angle $\alpha_3$. From angle $\alpha_3$, the edge will begin to grow until angle $\alpha_{5a}$, when the edge returns to a complete circle, and is no longer occluded by a limb. From here, we move to angle $\alpha_4$, where the limb of Sphere 1 will no longer be visible. This view will remain stable for the remainder of the event series.

In Intersection Case B (Figure 5.f), we first are capable of a view of the limb created by Sphere 2. We retain this view until we reach angle $\alpha_1$, when the limb of Sphere 2 first touches the limb of Sphere 1 (We note that one must reconstruct Figure 6 to find the proper series of angles and think in terms of intersected space.) Our view of one limb occluding that of the other remains until we reach angle $\alpha_{5a}$, where the edge first appears. From here we have a stable angle of the edge growing in size until we reach angle $\alpha_3$, when that edge appears as a line, with limbs at its borders. The edge will then continue to grow, while the limb created by Sphere 1 shrinks and the limb of Sphere 2 grows. When we reach angle $\alpha_{5a}$, the limb of Sphere 2 touches the limb of Sphere 1 at a point. Beyond that angle, we see the next stable view of a full circular limb from Sphere 2, and an almost completely circular edge, with the exception of a small portion of the edge occluded by the limb of Sphere 1. When we reach angle $\alpha_{5a}$, the edge will be at the point of occlusion, and beyond that angle, we see an entire edge in circular form inside of the limb of Sphere 2. This view will remain stable for the rest of the event series.

**Finding the Aspect Graph**

Having taken into consideration the above six event series, we must complete the proper angles so that we may complete the goal of creating the full Aspect Graph. First we write the equations of the spheres in two dimensional form (equations of circles, since we are partitioning longitudinal bands along the Gaussian Sphere this is valid assumption.) Referring to diagram 7, the equation of Sphere 1, with center located at $(X_{1a}, 0)$ where $X_{1a} = 0$, can then be written as $X_1^2 + Y_1^2 = r_1^2$ And that of Sphere 2, with center located at $(X_{2a}, 0)$ will be $(X_2 - X_{2a})^2 + Y_2^2 = r_2^2$ Solving for the set of simultaneous equations for intersections at the location $(X_{int}, Y_{int})$. We are interested only in the location $X_{int}$ on the X axis, for which we find the value to be

$$X_{int} = \frac{X_{2a} + r_1^2 - r_2^2}{2X_{2a}}$$

We first solve for angle $\alpha_{5a}$, as the tangent to Sphere 1 at point $X_{int}$. To obtain the tangent, we take the derivative of Sphere 1:

$$M_1 = -\frac{X_{int}}{\sqrt{r_1^2 - X_{int}^2}}$$

To find the angle with respect to the positive X axis, we need only take the arc tangent. Then solving for this angle with respect to the -X axis shows:

$$\alpha_{5a} = -\tan^{-1}(M_1)$$

62
Then, by inspection it is obvious that angle $\alpha_{2a}$ will be a complement of angle $\alpha_2$.

$$\angle a_{2a} = 180 - \angle a_2$$

Angle $\alpha_2$ is always 90 degrees since it is the line passing through a fixed set of points along $X_{int}$. We next solve for angle $a_4$. Taking the derivative of Sphere 2, and solving for the slope at $X_{int}$.

$$M_4 = \frac{(X_{int} - X_{2a})}{\sqrt{r_1^2 - (X_{int} - X_{2a})^2}}$$

Because angle $\alpha_4$ is capable of taking on both positive and negative values, depending on the location of $C_4$ with respect to the points of intersection, we define an always positive angle $\alpha_4$. The algorithm presented adjusts for cases when we should use either angle $a_4$, and/or angle $a_{4a}$. Thus if $M_4 < 0$ we use

$$\angle a_4 = 180 - \tan^{-1}(M_4)$$

and if $M_4 > 0$ we use

$$\angle a_4 = -\tan^{-1}(M_4)$$

Solving for $\angle a_{4a}$ as the complement of angle $a_4$. For the case when $M_4 < 0$ we use

$$\angle a_{4a} = -\tan^{-1}(M_4)$$

and if $M_4 > 0$ we use

$$\angle a_{4a} = 180 - \tan^{-1}(M_4)$$

Lastly we solve for angle $\alpha_1$, and its complement, angle $\alpha_{1a}$. We know that

$$Y_1' = Y_2' = m$$

(1), and labeling the point on Sphere 1 where the tangent is equal to $m$ as $(X_{11}, Y_{11})$. We then label the corresponding point on Sphere 2 as $(X_{22}, Y_{22})$. Another constraint can then be written as

$$m = \frac{Y_{11} - Y_{22}}{X_{11} - X_{22}}$$

(2). Since $Y_1 = f(X_1)$ and $Y_2 = f(X_2)$, equation 1 yields the form $f(X_1) = f(X_2)$, and equation 2 yields the form $\frac{Y_{11} - Y_{22}}{X_{11} - X_{22}}$, where $X_1 = X_{11}$ and $X_2 = X_{22}$. Solving the equalities for $X_{22}$, and substituting into $Y_2'$ we get a solution for the tangent $M_1$ to both spheres as

$$M_1 = \frac{r_1^2 - r_2^2}{\sqrt{r_1^2 - (Y_{22} - X_{22})^2}}$$

for which we calculate the angle $\alpha_1$ as

$$\angle a_1 = -\tan^{-1}(M_1)$$

To complete the Aspect Graph, we must take the computed angles and partition the Gaussian Sphere with longitudinal bands to form the regions of maximal stable viewspace. A representative view can be taken from within each range of space between bands of visual events to create the nodes of our aspect graph. The Aspect Graph consists of the unstable view obtained at the visual event. Having computed the above, we have completed the Aspect Graph of the object.

**DISCUSSION**

We have described a first step toward a general algorithm to compute the aspect graph of CSG objects. Using the Gaussian Sphere model of viewpoint space, we have shown how to compute the aspect graph of objects defined as the CSG of two spheres. We hope to extend this to more general subclasses of CSG objects. Our long-range goal will be reached through a series of steps to generalize the algorithm to handle interactions of multiple spheres, and then further extending into perspective projection. We feel that the aspect graph of such objects provides an accurate, comprehensive account of the possible views of an object based on its geometry, and will be pertinent in computer vision applications.

**REFERENCES**


Samuel, N.M., Requicha, A.A.G., and Elkaim, S.A. Methodology and Results of an Industrial Part Survey, Technical Memo 21, *Production Automation Project, University of*


To the Left of the Great Circle of S2 At the Great Circle of S2

To the Right of the Great Circle of S2

Figure 3

Stable Views

Union Case A
Fig. 5a

Union Case B
Fig. 5b

Difference Case A
Fig. 5c

Difference Case B
Fig. 5d

Intersection Case A
Fig. 5e

Intersection Case B
Fig. 5f

Figure 5

Figure 2

Figure 4

Figure 6

Figure 7

*Note, as the light sphere moves inward towards the left sphere, angle y1 will be less than 180 degrees.*
Using Saccadic Eye Movements to Create a 2.5-D Sketch

Katrina M. Simpson and Kevin W. Bowyer
Department of Computer Science and Engineering
University of South Florida
Tampa, Florida 33620 U.S.A.

(813) 974-3032
simpson@pluto.usf.edu, kwb@sol.usf.edu

Abstract. Computation of the 2.5-D sketch of a scene is an important element in computer vision systems. A new method for computing the 2.5-D sketch is proposed. This method is inspired by the saccadic eye movements of human vision. The basic principles are similar to binocular stereo, but one rotating camera is used. Results of sensitivity studies are given which suggest that this method may be practical.

INTRODUCTION

A large amount of research in computer vision has focused on methods for computing the 2.5-D sketch of a scene. The 2.5-D sketch is a representation of the depth (or "shape") of the surfaces in view in the scene. It is "2.5-D" in that it is constructed before objects are recognized and so the full 3-D depth of the objects whose "front" surfaces are in view is not yet known. The 2.5-D sketch may, in principle, be computed in a number of different ways, as evidenced by the use of the phrase shapes from X to describe this general area of research. The problem is how to use one or more 2-D images to construct a (possibly approximate or sparsely sampled) representation of the depth and orientation of the surfaces in the scene.

Due to the large volume of relevant literature, and the space limitations here, we cannot attempt a thorough review. The 2.5-D sketch concept was popularized by Marr (1982), who argued for its importance in object recognition. Aloimonos and Brown (1988) give a recent discussion of many of the different shape "cues" (stereo, shading, contour, ... ) and how information from different cues might be combined to achieve greater robustness.

This work is supported by Air Force Office of Scientific Research grant AFOSR-89-0036 and National Science Foundation grants IRI-8817778 and IRI-8900796 (Research Experiences for Undergraduates).


SACCADIC EYE MOVEMENTS

One interesting and puzzling feature of the human visual system is the purpose of saccadic eye movements. The eye fixates on a point in the scene for approximately 150 to 200 milliseconds and then quickly rotates to fixate on a different point in the scene. The human visual system makes approximately four such movements each second. The purpose of the saccades is not entirely clear, and they may in fact serve several different purposes. Binford has suggested that saccades may "... serve the purpose of self-calibration of imperfect sensors" (Binford, 1981, p. 205). Ballard reviews some of the psychological literature related to saccadic movements and vision, and suggests that saccades may be used in a task-dependent manner to aid in the recognition of objects in a scene (Ballard, 1987). The possibility of the human visual system obtaining depth information about the scene from saccades was noted by Helmholtz in his classic Treatise on Physiological Optics (Southall, 1962). To our knowledge, no one has yet explored this possibility from the standpoint of computer vision. The purpose of this paper is to investigate how saccades may be used in computing the depth information for a 2.5-D sketch. More precisely, we are taking the saccadic movements of human vision as an inspiration to explore a new way of computing the 2.5-D sketch in machine vision. The determination of whether or to what extent human vision actually uses saccades for a similar purpose is a very difficult question which we do not attempt to answer here.

To simplify the analysis, we initially consider monocular (one-eyed) vision. A single fixate-saccade-fixate sequence provides two stable images. In a manner similar to shape from stereo, it is possible to compute the 3-D coordinates of a feature point in the scene from the 2-D coordinates of its appearance in each of the two images. However, whereas in shape from stereo the two images to be analyzed come from different cameras at the same point in time, in our situation the two images come from the same camera at (slightly) different points in time. We speculate that it is possible to exploit this situation to provide a simple and reliable solution to the correspondence problem (the pairing of feature points in the two images).

We propose the following model for processing during a fixate-saccade-fixate sequence. During a fixation, the image is stable for a long enough time to allow substantial
processing to occur. Feature points may be detected in the image and their 2-D image coordinates determined. The saccade consists of a rapid, predetermined camera movement. During the saccade, the image is not stable for a long enough time to allow substantial processing—new feature points cannot be detected. However, the direction of the saccade is known, and so the expected direction of displacement of feature points determined during the fixation is also known. Therefore it may very well be feasible to perform sufficient computation to track feature points from one fixation through the rapidly varying images during a saccade to the next fixation. In this way, the correspondence of feature points between the two fixation images is automatic. During the next fixation, the displacement of feature points between fixation images may be used to compute depth information about points in the scene. (The fixate-saccade-fixate sequence is actually a continuous cycle, and so during each fixation there would be processing both to compute depth information for feature points from the last fixation and to determine new fixations.) The next section describes in more detail how this computation may be carried out.

**Computing a 2-D Sketch**

First, consider the situation at the fixation just preceding a saccade. Using a typical perspective projection model, the focal point lies on the Z axis at point (0, 0, f) and the image plane is coincident with the XY-plane. A particular point of interest on some object in the scene, a scene feature point, has coordinates (X0, Y0, Z0) in this coordinate system. The projection of the scene feature point onto the image plane gives rise to an image feature point. The coordinates of this image feature point in the image plane are:

\[
X_f = \frac{X_0 f}{Z_0} \quad Y_f = \frac{Y_0 f}{Z_0}
\]

The scene feature point, the camera focal point and the image feature point all lie on the same line. Thus, given an observed feature point extracted from an image and a known camera focal point, it is possible to determine the line through the scene on which the scene feature point must lie.

Now consider what happens during a saccade. It is clear that the eye rotates so as to direct the line of sight in a new direction. What is important is that the eye does not rotate around the focal point. In the human visual system, the focal point is approximately 12.9 mm in front of the center of rotation (Southall, 1962). Thus the eye can be idealized as rotating around a point at the back of the eye. This is important because if the eye rotated around the focal point, then the line on which the scene feature point, camera focal point and image feature point lie would not change with rotation of the eye, and there would be no information to use to determine the coordinates of the scene feature point. However, with rotation around a point at the back of the eye, the position of the focal point changes as the eye rotates and so the line determined from the image feature point and the focal point in the second fixation is different from that determined in the first fixation. If the displacement of the focal point between fixations is known, then both lines may be determined and their point of intersection gives the coordinates of the scene feature point.

Assume that the saccade rotates the original coordinate system about the z-axis by a known angle γ and about the y-axis by a known angle β. (See Figure 1.) Thus the saccadic movement is assumed to be a rotation around the (0, 0, f) point of the image plane, rather than the focal point.) The second fixation defines its own coordinate system, analogous to that of the first fixation, with the focal point again considered to be at (0, 0, f), and with the scene feature point at (X0, Y0, Z0). The two coordinate systems from the two fixations are related by a rotation matrix determined by γ and β. The equation for the line defined by the image feature point (Xf, Yf) and the focal point in the second fixation may be transferred into the coordinate system of the first fixation by the rotation matrix. The two lines intersect at the scene feature point. (See Figure 2.) In this way, we can solve for the (X, Y, Z) coordinates of the scene feature point in the coordinate system of the first fixation. For example, the solution for the Z coordinate of the scene feature point is:

\[
Z_f = \frac{Y_f f \cos \Theta - Y_0 f \sin \Theta - B f - Y_0 f \cos \Theta}{Y_f f \sin \Theta + X_f f \cos \Theta - X_0 f \cos \Theta}
\]

where A = Y_0 f \sin \Theta, B = X_0 f \sin \Theta, and Z = Y_0 f \cos \Theta. The equations for Xf and Yf can also be determined in a straightforward manner, but result in similarly complicated expressions and are not given here.

In order to determine the range of conditions over which a saccade might provide enough information to determine the coordinates of a scene feature point, a simple idealized simulation was set up. The focal length was set to 1. The image plane size was set to 3 focal lengths by 3 focal lengths, and discretized into 512 by 512 pixels (Table 1). (The screen size was set equal to 3 focal lengths because the size of the retina is roughly between 2 and 3 focal lengths.) For each postulated scene feature point location, the angle γ was varied in five degree increments from 0 degrees to 45 degrees. In order to simplify the equations for the simulation, the scene feature point was assumed to lie at some depth along the Z axis of the first fixation. The depth of the scene feature point was varied from 2 to 2800 units of the focal length, f. The results of the simulation experiment are summarized in Table 2. As the distance of the scene feature point from the image plane increases, the displacement between the image feature points decreases, as beta and gamma are kept constant. The displacements also increase as beta and gamma are increased, in which gamma has a larger influence because we are looking at the displacement in the Y direction. While the results of this simple idealized simulation suggest that a great deal of depth information can be computed over a wide range of depths, practical application will be limited by factors such as error in the measurement of the location of image
feature points, camera rotation angles, and other factors.

In order to determine important variables in the equation which would have an effect on camera design, a second simulation was conducted. The question was, what kind of error would result if the motors controlling a rotating camera were only accurate to plus or minus a degree. The difference between the Z distance values calculated for plus and minus one degree of rotation about the x-axis (gamma) was called the error window. The greater the distance of the object from the camera, the greater the error window (figure 3). The error window also increased with smaller focal lengths (figure 4). The error caused by digitization of the image was examined by looking at different image resolutions. The resolution or pixel size made little difference for larger (> 20 degrees) rotations (figure 5). In these graphs, one can also notice the general decreasing of the error window size as gamma increases.

**DISCUSSION**

The results of the idealized simulation experiment show that saccades do have the potential to provide sufficient information to allow the construction of a 2-D sketch of the "near" surfaces in a scene. We are now working on creating a "saccadic" camera system and designing a set of experiments with real image data, in order to better determine the practical utility of this approach. The full paper will describe the "saccadic" camera set-up and the results of our initial experiments with real image data.

**REFERENCES**


<table>
<thead>
<tr>
<th>SCREEN SIZE</th>
<th>3 x 3 *</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIKELS PER SCREEN</td>
<td>512 x 512</td>
</tr>
<tr>
<td>RANGE OF BETA</td>
<td>-45° to 45°</td>
</tr>
<tr>
<td>RANGE OF GAMMA</td>
<td>-45° to 45°</td>
</tr>
</tbody>
</table>

* in units of focal length

Table 1—Values used in mathematical test of limits of use

---

**Figure 1—Illustration of a rotation of the coordinate system during a saccade.**

**Figure 2—Two lines of sight used to determine (x,y,z) of an object point.**
### TABLE 2 - Values of \( Y \) of the Second Image Point in Units of Focal Length as \( Z \) varies

#### z=2 focal lengths

<table>
<thead>
<tr>
<th>beta</th>
<th>0°</th>
<th>5°</th>
<th>10°</th>
<th>15°</th>
<th>20°</th>
<th>25°</th>
<th>30°</th>
<th>35°</th>
<th>40°</th>
<th>45°</th>
</tr>
</thead>
<tbody>
<tr>
<td>gamma</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0°</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>5°</td>
<td>0.1756</td>
<td>0.1763</td>
<td>0.1764</td>
<td>0.1821</td>
<td>0.1878</td>
<td>0.1961</td>
<td>0.2081</td>
<td>0.2259</td>
<td>0.2527</td>
<td>0.2915</td>
</tr>
<tr>
<td>10°</td>
<td>0.3752</td>
<td>0.3796</td>
<td>0.3840</td>
<td>0.3771</td>
<td>0.3836</td>
<td>0.4009</td>
<td>0.4292</td>
<td>0.4638</td>
<td>0.5229</td>
<td>0.5233</td>
</tr>
<tr>
<td>15°</td>
<td>0.5255</td>
<td>0.5378</td>
<td>0.5446</td>
<td>0.5774</td>
<td>0.5968</td>
<td>0.6248</td>
<td>0.6511</td>
<td>0.7289</td>
<td>0.8263</td>
<td>1.0090</td>
</tr>
<tr>
<td>20°</td>
<td>0.7779</td>
<td>0.7813</td>
<td>0.7818</td>
<td>0.8104</td>
<td>0.8391</td>
<td>0.8815</td>
<td>0.9499</td>
<td>1.0366</td>
<td>1.1618</td>
<td>1.4705</td>
</tr>
<tr>
<td>25°</td>
<td>1.0401</td>
<td>1.0451</td>
<td>1.0503</td>
<td>1.0873</td>
<td>1.1722</td>
<td>1.1918</td>
<td>1.2847</td>
<td>1.4282</td>
<td>1.6665</td>
<td>2.1216</td>
</tr>
<tr>
<td>30°</td>
<td>1.3600</td>
<td>1.3732</td>
<td>1.3954</td>
<td>1.4352</td>
<td>1.4973</td>
<td>1.5074</td>
<td>1.5721</td>
<td>1.6555</td>
<td>2.3439</td>
<td>3.1483</td>
</tr>
<tr>
<td>35°</td>
<td>1.7072</td>
<td>1.8080</td>
<td>1.8417</td>
<td>1.9023</td>
<td>1.9981</td>
<td>2.1440</td>
<td>2.3721</td>
<td>2.7475</td>
<td>3.4460</td>
<td>5.1193</td>
</tr>
<tr>
<td>40°</td>
<td>2.1611</td>
<td>2.2326</td>
<td>2.4092</td>
<td>2.7877</td>
<td>3.9772</td>
<td>5.9826</td>
<td>9.1256</td>
<td>1.6714</td>
<td>2.9067</td>
<td></td>
</tr>
<tr>
<td>45°</td>
<td>3.1412</td>
<td>3.4460</td>
<td>3.5463</td>
<td>3.7321</td>
<td>4.8902</td>
<td>5.4598</td>
<td>5.4496</td>
<td>7.3111</td>
<td>12.0081</td>
<td></td>
</tr>
</tbody>
</table>

#### z=602 focal lengths

<table>
<thead>
<tr>
<th>beta</th>
<th>0°</th>
<th>5°</th>
<th>10°</th>
<th>15°</th>
<th>20°</th>
<th>25°</th>
<th>30°</th>
<th>35°</th>
<th>40°</th>
<th>45°</th>
</tr>
</thead>
<tbody>
<tr>
<td>gamma</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0°</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>5°</td>
<td>0.0876</td>
<td>0.0874</td>
<td>0.0876</td>
<td>0.0876</td>
<td>0.0876</td>
<td>0.0877</td>
<td>0.0877</td>
<td>0.0877</td>
<td>0.0877</td>
<td>0.0877</td>
</tr>
<tr>
<td>10°</td>
<td>0.1766</td>
<td>0.1766</td>
<td>0.1766</td>
<td>0.1766</td>
<td>0.1767</td>
<td>0.1767</td>
<td>0.1767</td>
<td>0.1767</td>
<td>0.1767</td>
<td>0.1767</td>
</tr>
<tr>
<td>15°</td>
<td>0.2654</td>
<td>0.2654</td>
<td>0.2654</td>
<td>0.2654</td>
<td>0.2655</td>
<td>0.2655</td>
<td>0.2655</td>
<td>0.2655</td>
<td>0.2655</td>
<td>0.2655</td>
</tr>
<tr>
<td>20°</td>
<td>0.3546</td>
<td>0.3546</td>
<td>0.3546</td>
<td>0.3546</td>
<td>0.3546</td>
<td>0.3546</td>
<td>0.3546</td>
<td>0.3546</td>
<td>0.3546</td>
<td>0.3546</td>
</tr>
<tr>
<td>25°</td>
<td>0.4672</td>
<td>0.4672</td>
<td>0.4672</td>
<td>0.4672</td>
<td>0.4672</td>
<td>0.4672</td>
<td>0.4672</td>
<td>0.4672</td>
<td>0.4672</td>
<td>0.4672</td>
</tr>
<tr>
<td>30°</td>
<td>0.5765</td>
<td>0.5785</td>
<td>0.5785</td>
<td>0.5785</td>
<td>0.5785</td>
<td>0.5785</td>
<td>0.5785</td>
<td>0.5785</td>
<td>0.5785</td>
<td>0.5785</td>
</tr>
<tr>
<td>35°</td>
<td>0.7010</td>
<td>0.7016</td>
<td>0.7017</td>
<td>0.7017</td>
<td>0.7017</td>
<td>0.7017</td>
<td>0.7017</td>
<td>0.7017</td>
<td>0.7017</td>
<td>0.7017</td>
</tr>
<tr>
<td>40°</td>
<td>0.8409</td>
<td>0.8409</td>
<td>0.8410</td>
<td>0.8410</td>
<td>0.8410</td>
<td>0.8410</td>
<td>0.8410</td>
<td>0.8410</td>
<td>0.8410</td>
<td>0.8410</td>
</tr>
<tr>
<td>45°</td>
<td>1.0024</td>
<td>1.0024</td>
<td>1.0024</td>
<td>1.0024</td>
<td>1.0025</td>
<td>1.0025</td>
<td>1.0025</td>
<td>1.0025</td>
<td>1.0025</td>
<td>1.0025</td>
</tr>
</tbody>
</table>

#### z=1002 focal lengths

<table>
<thead>
<tr>
<th>beta</th>
<th>0°</th>
<th>5°</th>
<th>10°</th>
<th>15°</th>
<th>20°</th>
<th>25°</th>
<th>30°</th>
<th>35°</th>
<th>40°</th>
<th>45°</th>
</tr>
</thead>
<tbody>
<tr>
<td>gamma</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0°</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>5°</td>
<td>0.0375</td>
<td>0.0376</td>
<td>0.0376</td>
<td>0.0376</td>
<td>0.0376</td>
<td>0.0376</td>
<td>0.0376</td>
<td>0.0376</td>
<td>0.0376</td>
<td>0.0376</td>
</tr>
<tr>
<td>10°</td>
<td>0.1785</td>
<td>0.1785</td>
<td>0.1785</td>
<td>0.1785</td>
<td>0.1785</td>
<td>0.1785</td>
<td>0.1785</td>
<td>0.1785</td>
<td>0.1785</td>
<td>0.1785</td>
</tr>
<tr>
<td>15°</td>
<td>0.2682</td>
<td>0.2682</td>
<td>0.2682</td>
<td>0.2682</td>
<td>0.2682</td>
<td>0.2682</td>
<td>0.2682</td>
<td>0.2682</td>
<td>0.2682</td>
<td>0.2682</td>
</tr>
<tr>
<td>20°</td>
<td>0.3544</td>
<td>0.3544</td>
<td>0.3544</td>
<td>0.3544</td>
<td>0.3544</td>
<td>0.3544</td>
<td>0.3544</td>
<td>0.3544</td>
<td>0.3544</td>
<td>0.3544</td>
</tr>
<tr>
<td>25°</td>
<td>0.4663</td>
<td>0.4663</td>
<td>0.4663</td>
<td>0.4663</td>
<td>0.4663</td>
<td>0.4663</td>
<td>0.4663</td>
<td>0.4663</td>
<td>0.4663</td>
<td>0.4663</td>
</tr>
<tr>
<td>30°</td>
<td>0.5769</td>
<td>0.5790</td>
<td>0.5790</td>
<td>0.5790</td>
<td>0.5790</td>
<td>0.5790</td>
<td>0.5790</td>
<td>0.5790</td>
<td>0.5790</td>
<td>0.5790</td>
</tr>
<tr>
<td>35°</td>
<td>0.7011</td>
<td>0.7011</td>
<td>0.7011</td>
<td>0.7011</td>
<td>0.7011</td>
<td>0.7011</td>
<td>0.7011</td>
<td>0.7011</td>
<td>0.7011</td>
<td>0.7011</td>
</tr>
<tr>
<td>40°</td>
<td>0.8402</td>
<td>0.8402</td>
<td>0.8402</td>
<td>0.8402</td>
<td>0.8402</td>
<td>0.8402</td>
<td>0.8402</td>
<td>0.8402</td>
<td>0.8402</td>
<td>0.8402</td>
</tr>
<tr>
<td>45°</td>
<td>1.0014</td>
<td>1.0014</td>
<td>1.0014</td>
<td>1.0025</td>
<td>1.0025</td>
<td>1.0025</td>
<td>1.0025</td>
<td>1.0025</td>
<td>1.0025</td>
<td>1.0025</td>
</tr>
</tbody>
</table>
Error window vs gamma as depth changes

- Column 2: z=2
- Column 4: z=10

Motor accuracy: 1 degree

Beta: 0

Focal length: 1

X=0

Y=0

Resolution: 512 X 512

Figure 3

Error window vs gamma as focal length changes

- Column 2: f=1
- Column 4: f=5
- Column 6: f=7

Motor accuracy: 1 degree

Beta: 0

X=0

Y=0

Z=10

Resolution: 512 X 512

Figure 4

Error window vs gamma as resolution changes

- Column 2: resolution=512
- Column 4: resolution=128
- Column 6: resolution=256
- Column 8: resolution=1024

Motor accuracy: 1 degree

Beta: 0

Focal length: 1

X=0

Y=0

Z=10

Figure 5
REPRESENTING THE VISUAL POTENTIAL OF A NON-RIGID ASSEMBLY OF RIGID PARTS

Maha Sallam and Kevin Bowyer
Department of Computer Science and Engineering
University of South Florida
Tampa, Florida 33620 U.S.A.
sallam@pluto.usf.edu, kwb@sol.usf.edu

Abstract.
The aspect graph is an important object representation for use in computer vision. A number of algorithms have been presented to compute the aspect graph of various classes of rigid objects. We present a generalization of the aspect graph concept to include objects defined as non-rigid assemblies of rigid parts.

INTRODUCTION

Finding an efficient way of representing knowledge about the visual appearance of objects is an important step in the process of devising many object recognition algorithms. The aspect graph is a means of representing knowledge about all the possible fundamentally different views of an object. In the past few years, a number of researchers have devised algorithms to compute the aspect graph for different classes of objects under different models of viewpoint space. The two basic models of viewpoint space that have been used are: 1) the Gaussian sphere, for orthographic projection (Gigus and Malik, 1988; Gigus et al, 1988), and 2) all of 3-D space, for perspective projection, (Stewman, 1988; Plantinga, 1987). Different classes of objects that have been considered are 1) convex polyhedra, 2) general polyhedra (Gigus and Malik, 1988; Gigus et al, 1988; Stewman, 1988; Plantinga, 1987), and 3) solids of revolution (Egbert, 1989; Kriegman, 1989).

Many techniques exist which allow an object to be modeled as a rigid assembly of rigid parts, but very few schemes have been proposed which allow the modeling of non-rigid connections between parts of an object. A recent example of such a modeling scheme, limited to 2-D objects, has been described by Grimson (1989). To date, all work concerned with computing the aspect graph of an object has assumed that the object is entirely rigid.

The purpose of this report is to describe a generalization of the aspect graph concept which leads to a representation for the visual potential of 3-D objects modeled as non-rigid assemblies of rigid parts. The next section of this paper defines the aspect graph concept in greater detail. Then we present a method of modeling objects as non-rigid assemblies of rigid parts. Finally, we describe a generalization of the aspect graph concept which can be used to create a representation of the visual potential of non-rigid assemblies.

ASPECT GRAPHS

The idea of aspect graphs was first introduced by Koen- derink and van Doorn (1979). In general, an aspect graph is a structure that defines the set of all qualitatively different 2-D views of an object as seen from all points in some viewpoint space. These views can be general or accidental. For each general view, a non-zero volume cell of the viewing space can be found, such that this particular view is seen from any point in the cell. An accidental view, which we will call a visual event, is only a transitional view between two general views and can only be seen when the viewer is on a boundary between general-view cells. The aspect graph also describes the adjacencies of all viewing cells.

More formally then, an aspect graph can be defined as follows. It is a way of combining all information obtained from different 2-D views of an object into one complete description of the 3-D object by using a graph in which:

- each node represents a topologically distinct view and a viewing cell of space from which this view is seen, and

- each arc corresponds to an accidental view or an event and represents the relationship between two
nodes or the adjacency of the viewing cells represented by the nodes.

As an example, consider the object illustrated in Figure 1. The aspect graph for this object can be computed (using an appropriate algorithm from (Gigué and Malik, 1988; Gigué et al, 1989; Plantings, 1987; Stewman, 1988) by considering all the planes in which the faces of the object lie as well as all the planes defined by the edge-vertex pairs of the object. Examples of these planes defined by edge-vertex interaction are depicted in Figure 2.

NON-RIGID ASSEMBLIES OF RIGID PARTS

In order to be able to model jointed assemblies of rigid parts, we first need to rigorously define the class of objects. We define a non-rigid assembly as an object:

- that is composed of two or more individual rigid parts (for the moment, we assume that each part is modeled as a polyhedra),
- where each part is connected to one or more of the other parts at all times, and
- where each connection can allow: 1) no movement, 2) translational movement, 3) rotational movement or 4) a combination of translational and rotational movement.

Translational type connections are connections which allow translation of one part of the object relative to another. In the simplest case, a 1-D translational connection, one part may translate relative to another along a line segment. These are the least complex types of non-rigid connections since they allow only one degree of freedom and the equations of displacement for the moving part are easy to calculate. More complex translational connections may allow translation throughout a 2-D surface patch or a 3-D volume.

Rotational type connections are connections that allow rotation of one part of the object about a fixed axis relative to another part of the same object. In the simplest case, there is only a single axis of rotation and an allowed range of rotation values around that axis. Like a 1-D translational connection, these connections have only one degree of freedom, but the equations for displacement are not as easy to calculate. More complex rotational connections may combine multiple axes of rotation (for example, to get a chair in which you can turn around and lean back).

For all the types of connections described, an initial position of the moving part of the assembly and a range of movement must be given as a part of the object description. As an example of an assembly, consider the object depicted in Figure 3. This object appears similar to the rigid object in Figure 1, but we now consider it as an assembly composed of two rigid parts, a base and an appendage, which may slide relative to one another. So this object is a jointed assembly of two rigid parts with a 1-D translational connection between them. We can consider the base to be the fixed part of the object and the appendage to be the moving part. The appendage has a range of sliding that can produce the various assembly configurations illustrated in Figure 4.

THE VISUAL POTENTIAL OF ASSEMBLIES

The idea of using aspect graphs to represent rigid objects can be generalized to include non-rigid assemblies of rigid parts. However, what makes this generalization very difficult is the fact that we need to consider not only the infinite number of possible 2-D views of each configuration of the assembly but also we need to somehow represent the infinite number of configurations that the assembly may assume. Each of these object configurations can be viewed as a rigid object for which an aspect graph can be calculated. This solves the problem of representing the infinite number of views for each configuration, but it also produces an infinite number of aspect graphs that must be reduced to a finite number if they are to be used for modeling the assembly.

Just as we classified the possible views of a rigid object into general views and accidental views in order to subdivide a viewpoint space and compute the aspect graph, we can classify the possible configurations of a non-rigid assembly as general configurations and accidental configurations to subdivide a connection parameter space and compute the visual potential of the assembly. The connection parameter space is a space of dimension equal to the total number of degrees of freedom in all the connections in the particular assembly that it represents. So, for example, a completely rigid object such as that depicted in Figure 1 would have a 0-D connection parameter space, and the object depicted in Figure 3 would have a 1-D connection parameter space representing its idealized translational connection.

The desired subdivision of the connection parameter space is defined as follows.

- A point in the connection parameter space represents a general configuration if, for all directions in the connection parameter space, an infinitesimal change in that direction results in a configuration whose aspect graph is isomorphic to that of the original configuration.
- A point in the connection parameter space represents an accidental configuration if there is any direction in which an infinitesimal change results in a configuration whose aspect graph is not isomorphic to the original configuration.
- Each cell of the connection parameter space created by the subdivision is a maximal connected region representing general configurations of the assembly.
- The boundaries of the subdivision correspond to connection parameter values which represent accidental configurations. These accidental configurations are transitions between ranges of general configuration.
Table 1.  
Face vertices

<table>
<thead>
<tr>
<th>face</th>
<th>vertices</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>(5, 6, 2, 1)</td>
</tr>
<tr>
<td>b</td>
<td>(1, 2, 3, 4)</td>
</tr>
<tr>
<td>c</td>
<td>(4, 3, 7, 8)</td>
</tr>
<tr>
<td>d</td>
<td>(8, 7, 6, 3)</td>
</tr>
<tr>
<td>e</td>
<td>(2, 6, 7, 3)</td>
</tr>
<tr>
<td>f</td>
<td>(5, 1, 4, 8), (10, 14, 15, 11)</td>
</tr>
<tr>
<td>g</td>
<td>(13, 14, 10, 9)</td>
</tr>
<tr>
<td>h</td>
<td>(9, 10, 11, 12)</td>
</tr>
<tr>
<td>i</td>
<td>(12, 11, 15, 16)</td>
</tr>
<tr>
<td>j</td>
<td>(16, 13, 14, 13)</td>
</tr>
<tr>
<td>k</td>
<td>(13, 9, 12, 16)</td>
</tr>
</tbody>
</table>

Table 2.  
Vertex x y z

<table>
<thead>
<tr>
<th>vertex</th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>1.5</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>1.5</td>
<td>2</td>
</tr>
<tr>
<td>11</td>
<td>3</td>
<td>1.5</td>
<td>2</td>
</tr>
<tr>
<td>12</td>
<td>3</td>
<td>1.5</td>
<td>3</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>0.5</td>
<td>3</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
<td>0.5</td>
<td>2</td>
</tr>
<tr>
<td>15</td>
<td>3</td>
<td>0.5</td>
<td>2</td>
</tr>
<tr>
<td>16</td>
<td>3</td>
<td>0.5</td>
<td>3</td>
</tr>
</tbody>
</table>

Figure 1.  
An example of a rigid object.

Figure 2.  
Example of an auxiliary plane needed to calculate the boundaries of the viewing space cells for the rigid object in figure 1.
Each accidental configuration of the assembly can be considered as a rigid object with rigid connections whose aspect graph can be calculated independently. And for each group of general configurations described by general parameter values that belong to one cell of the connection parameter space, one aspect graph can be calculated by taking one representative configuration of the subdivision and calculating its aspect graph as if it were a single rigid object.

Consider the jointed assembly illustrated in Figure 3. This object has three general configurations depicted in Figures 4a, 4c, and 4e, and two accidental configurations depicted in Figures 4b and 4d. The connection parameter space, as indicated earlier, is a 1-D space representing the displacement of the appendage relative to the base, and consists, in this case, of three different 1-D cells and two 0-D boundaries between them. One of the general configurations contains a particular configuration which corresponds to the object in Figure 1.

SUMMARY
This paper presents a generalization of the aspect graph concept which can be used to define the complete visual potential of objects which are represented as a non-rigid assembly of rigid parts. All previous work with the aspect graph concept has assumed that the object is entirely rigid. Thus this work is an important step forward in knowledge representation for computer vision.

REFERENCES


73
<table>
<thead>
<tr>
<th>vertex</th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

**base**

<table>
<thead>
<tr>
<th>vertex</th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>x</td>
<td>1.5</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>x</td>
<td>1.5</td>
<td>2</td>
</tr>
<tr>
<td>11</td>
<td>x+2</td>
<td>1.5</td>
<td>2</td>
</tr>
<tr>
<td>12</td>
<td>x+2</td>
<td>1.5</td>
<td>3</td>
</tr>
<tr>
<td>13</td>
<td>x</td>
<td>0.5</td>
<td>3</td>
</tr>
<tr>
<td>14</td>
<td>x</td>
<td>0.5</td>
<td>2</td>
</tr>
<tr>
<td>15</td>
<td>x+2</td>
<td>0.5</td>
<td>2</td>
</tr>
<tr>
<td>16</td>
<td>x+2</td>
<td>0.5</td>
<td>3</td>
</tr>
</tbody>
</table>

**appendage**

**Connection:**

Parts: appendage, base.
Type: linear translation along L
Connection Point: Q (x+1, 1, 2).
Connection Range: [ x=-1 to x=3 ]

Figure 3
Complete description of a jointed assembly of rigid parts

(a)  (b)  (c)  (d)  (e)

Figure 4.
Various configurations that the assembly in figure 4 may assume as seen from the side
STEREO PROCESSING WITH ARTIFICIAL NEURAL NETWORKS

Thomas L. Clarke
Institute for Simulation and Training
and Mathematics Department
University of Central Florida
12424 Research Parkway, Suite 300
Orlando, FL 32826

ABSTRACT

Mission planning, rehearsal and training systems require terrain databases based on the most current available intelligence. Present systems for producing terrain databases are both time and resource intensive. As a result, many automated decision aids that require terrain-specific data are not now practical.

Research into using a neural network system for rapidly producing terrain databases from pairs of stereo photographs has been promising. A feedforward architecture patterned after the vision centers of the brain has been found to be able to recognize random dot stereograms. Performance estimates show that the neural network based system will be able to process stereograms as rapidly as more conventional approaches.

INTRODUCTION

Background

Terrain data is the basic input to many planning and training systems, both military and civilian. Planning requires terrain data to provide the background against which plans are made. Training simulators make extensive use of terrain data to provide the visual data base against which the simulator appears to move. While there is a large amount of terrain and Geographical Information System (GIS) data available from agencies such as the Defense Mapping Agency (DMA), and the US Geological Survey (USGS), most of this data is several years old. This archived data does not reflect recent anthropomorphic and natural changes to terrain and landscape. This data can be brought up to date by incorporating the

---

Fig 1. Random dot stereogram. To view this random dot stereogram, unfocus your eyes and lower your head so that a row of three squares is visible. After a few seconds processing delay a square will appear to rise above the background of the middle square as suggested in the lower portion of the figure.
The vision systems of the brain are able to process stereo image pairs with
great facility. The reader can verify this by using his vision system to view
the random-dot stereograms shown in figure 1. Individually each of the two
images in figure 1 is merely a random array of black and white pixels. Only
when the two are processed as a stereo pair does the elevated square appear.
The ability of the visual system to extract this information from two noise-
like images is quite remarkable.

Approaches to Stereo

Two approaches to processing stereo imagery are commonly used. One is based
on statistically matching local patches of imagery. The second relies on
matching significant features in the two images. The patch method is commonly
used in terrain processing, whereas the feature matching approach is commonly
used in machine vision.

The approach reported by (Hannah 1988) uses correlation of both types of
image feature. The approach of (Boulou and Chen 1988) combines feature matching
with image reconstruction. In effect they combine the stereo processing with
image synthesis and the use a "reasonedness" measure on the resulting
image to assist in matching the images.

The availability of highly parallel computers like the Connection Machine has
motivated the use of algorithms that, while computationally intensive, lend
themselves to parallelization. Examples of these algorithms are found in (Barnard
1988) who used a simulated annealing approach in analogy to statistical
mechanics. A similar approach is taken by (Stewart and Dyer 1988) who use a
connectionist or parallel architecture to solve a system of constraint equations
that the two images must satisfy.

The parallel or connectionist approaches outlined above are similar in
some respects to the system investigated here. Neural networks, simulated
annealing and connectionist constraint resolution all involve interactive
processing among what can be thought of as an array of many small processors.

A distinguishing feature of the present approach is that it is modeled
after systems used in the brain, so that the optimization of millennia of
evolution can be utilized. An animal with an inefficient stereo processing
capability would not have survived natural selection.

Recent studies (Levay et al 1985) have shown that in higher animals the
visual fields of the left and right eyes are not mapped onto separate areas of
the digital cortex, but are mapped in the form of narrow, essentially vertical
strips. These strips alternate, left

eye, right eye... It is easy to see how
this architecture facilitates correlation
of small patches of visual field as in
the statistical approach. For the small
values of image shift caused by stereo
disparity, the alternate strip mapping
will place regions of the left and right
images that originate from the same
physical location near each other.

A system for generating photo-
texture/terrain-elevation database from
stereo pairs based on a neural network
modeled after the architecture used in
the visual cortex of the brain is being
developed. The back propagation
algorithm (McClelland and Rumelhart 1986)
was used to train a feed forward network
to extract disparity information from
pairs of random dot stereograms. The
network should be able to "generalize"
from the random dot training images to
more realistically textured images.

DISCUSSION

Mathematics

The underlying mathematics of any
approach to stereo processing is that of
maximum-likelihood estimation (Duda
and Hart 1973). Using a simplified
notation, let \( I_1(u,v) \) and \( I_2(u,v) \) be the
observed stereo image pair; \( u \) and \( v \) are
image coordinates. We desire a terrain
elevation field \( S(x,y) \) and reflectance
field \( R(x,y) \) which give \( I_1 \) and \( I_2 \) when
remotely sensed; \( x \) and \( y \) are actual
position coordinates. If \( S_1 \) and \( S_2 \) are
the sensing functionals which are
determined by viewing angle, sun-angle
and the like, then \( S \) and \( R \) should
satisfy:

\[
I_1 = S_1[Z,R], \quad I_2 = S_2[Z,R].
\]

In general \( S_1 \) and \( S_2 \) are very
complicated functionals, since \( I_1 \) and \( I_2 \)
depend not only on \( R \) at the corresponding
position but on the surface slope and
hence the derivatives of \( Z \). We will
assume that the derivatives are smooth so
that \( S_1 \) can be treated as a functional of
\( R \) with \( Z \) as a parameter. In this case
\( S_1[u,v] \) is proportional to \( I_1[g_1(x,y,z)] \) where
\( g_1 \) is the geometric transformation
from \((x,y,z)\)-space to \((u,v)\)-space for the
ith image. Invoking proportionality
covers a multitude of sins, of course,
but since correlations are used below the
absolute proportionality constants drop
out.

If \( I_1 \) and \( I_2 \) are well-behaved
(there lies a tale, checkerboard fields
could be a problem), a small patch of $I_1$ centered at $(u_1, v_1)$ will be found to have maximum correlation with the small patch of $I_2$ centered at $(u_2, v_2)$. It is then straightforward to solve the two algebraic equations

\[ (u_1, v_1) = g_1(x, y, z) \quad \text{and} \quad (u_2, v_2) = g_2(x, y, z), \]

for $x$, $y$, and $z$. $R$ at this point $(x, y)$ is obtained by applying radiometry to observed image intensities. Repeating this process for every pixel leads to a complete solution for the $R$ and $Z$ fields. Provided the usual statistical assumptions are valid, this will be a maximum-likelihood estimate.

**Implementation**

As discussed above this process of correlation and maximum likelihood estimation has been implemented in various ways. Using an artificial neural network has many advantages. The use of neural network techniques couples the correlation/estimation process for every pixel in the image so that a globally optimal estimate can be formed. The correlation/estimation step can be implemented in the form of an artificial neural network. This approach has many potential advantages:

1. The adaptive learning properties of neural networks will permit the estimation parameters to adjust and produce the better depth estimate, allowing for factors that are not easily incorporated into formal mathematical models,

2. The property that neural networks have of degrading gracefully when given noisy inputs will ensure that the depth data produced will be useful even when the input data are marginal, and

3. The ability of higher animals to extract depth information from stereo pairs shows that this approach will work for a wide variety of input images.

Figure 2 shows how this works in the proposed processing system. The phototexture features lie at different locations on each of the two stereo photos. The photos are divided into strips or decimated and then conceptually reassembled into a composite image which serves as the input to an artificial neural-network (ANN) which performs the correlation and estimation to generate terrain elevation data. Once the terrain elevation data is available it is a simple matter to use the geometric and photometric transformation equations to generate a photo-texture/terrain-elevation data base.

The neural network used has been organized in a simple feed forward architecture (Lippman 1987). The image pixels are input to a layer of neurons which in turn feed another layer and so on. There are no interconnections between the neurons within a layer, or from a lower layer to a higher layer. The computation time of such a network is thus a polynomial power of the number of input pixels, $N$. The polynomial power is determined by the number of layers, $M$. The polynomial constants are determined by the fraction $a$ of non-zero inputs at each layer and the number of operations $b$ required to compute the neurons sigmoidal function; on average a neuron receives an non-zero weights from the layer above. Under these simplifying assumptions, the feedforward network can compute its result in order $(aN^{(M-1)} + bN^M$ operations.

Other neural network architectures such as Boltzman, adaptive resonance, and Hopfield include feedback connections within a layer and upward between layers. As such the time required to complete a calculation is not easily predictable. The feedback connections introduce a potential oscillatory component into the network output which must be allowed to dampen out before the output is sampled. Nevertheless, feedback connections may have use in enforcing desired smoothness.
RESULTS

Feasibility

The feasibility of this approach has been demonstrated by generating a set of random dot stereograms as shown in figure 3. These 64 pixel by 64 pixel stereograms were input as pairs of raster scan lines to a three-layer feedforward neural network \( (M = 3) \), with equal numbers of neurons in each layer \( (N = 64) \) using the commercial SAIC ANSim (Artificial Neural Simulator) package.

The training set was limited to the stereograms with 1, 2, and 4 raised/lowered sub-squares (6 stereograms total); the stereograms with 8 squares were reserved to test the ability of the network to generalize. The initial RMS output error was .21; note that the sub-square was raised/lowered by \( +/- .5 \) units, so that .21 corresponds to random output. The results of training using the back propagation algorithm are shown in the table. The training process began to approach an asymptotic limit after about 200 cycles. It was possible to reduce the error further by introducing a new set of random texture (dots) into the stereograms. Watching the graphic display of neuron excitations provided by the ANSim package showed that the network was able to reliably identify the raised/lowered portions of the stereograms.

<table>
<thead>
<tr>
<th>Number of Training Cycles</th>
<th>RMS Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>.21</td>
</tr>
<tr>
<td>209</td>
<td>.041</td>
</tr>
<tr>
<td>374</td>
<td>.039</td>
</tr>
<tr>
<td>535</td>
<td>.029</td>
</tr>
</tbody>
</table>

When the 16 sub-square stereograms were presented to the trained network, the results were at first disappointing. The RMS error was .30 for the 16 sub-square images. It was soon determined that the network could quickly learn to recognize these finer images. After 60 additional training cycles using the 16 sub-square images the error dropped to .07. This quick learning of a new type of image after having learned other types of images suggests the investigation of hierarchical training schemes wherein the network is trained on one type of image at a time rather than the whole set of images at once.

The network was trained on an 386-type 16 MHz PC equipped with an 80387 numeric coprocessor. Nevertheless, a cycle through the entire 6 image (184 raster-line pairs) training set took 6 minutes. Hence the large training runs reported above were conducted overnight. The ability of the simple feedforward network to learn to recognize these random-dot stereograms shows that the approach proposed here is viable.

Performance

The anticipated performance of the stereo processing system in an operational mode can be estimated on the basis of the above results. The key observation is that the weight matrix of the network trained as described above was very sparse. That is, the input weighting values for the artificial neurons were mostly zero. On average the neurons in the network trained above received non-zero inputs from only about ten percent of the neurons in the layer above. The fraction \( a \) is thus about 1/10.

For a three-layer system processing a 1024 by 1024 pixel image raster-scan fashion a above, the number of multiplications per image is thus about \( 2 \times 2048 \times 205 = 839,680 \). That is the one less than the number of layers multiplied by the product of twice the number of pixel pairs per raster scan and the number of non-zero input weights to each neuron.

Fig 3. Random dot stereograms used to train the demonstration neural network. The darkened sub-squares show the regions in which the output is higher/lower than the background.
Thus an estimate of a million floating point multipies and accumulates to process each 1024 pixel raster scan row is reasonable. This estimate is likely very generous as the value of 206 non-zero weights seems rather high. The total time to process the whole 1024 by 1024 pixel image array would thus be about 1000 seconds on a 1 MFLOP (million floating point operations per second) machine such as an coprocessor equipped 386-type PC.

Published performance figures are rare since stereo image processing commonly depends on interactive human intervention so that the machine processing time is not the ultimate limiting factor. Nevertheless the above estimate compares favorably to the 1300 seconds that would be required to process a 1024 square pixel image using simulated annealing on a Symbolics 3600 (Barnard 1988).

Less than twenty minutes to process an image pair is not an excessive delay to receive an operationally useful up-to-date terrain data set from aerial or satellite reconnaissance. Note that Digital Signal Processor (DSP) and Reduced Instruction Set Computer (RISC) based coprocessors are now available rated at 30 MFLOPS or more, so that the stereo image processing time would drop to under a minute in a practical system.

Conclusions

Terrain data and associated aerial or satellite photography needed for training and planning form very large data sets. Any method for automating the merging or production of combined photo-texture and terrain data sets must efficiently handle very large amounts of data. The artificial neural network based method reported here is patterned after higher-animal nervous systems and promises to provide that efficiency.

The computations performed by this feed forward stereo processing system are an example of the ability of neural networks to process a large class of problems in log time. It thus appears likely that the stereo processing problem is of the class C of problems solvable by a network of processors whose depth is logarithmic in the problem size (Balcazar et al 1988). Simulation of a parallel neural network on a serial computer leads, of course, to polynomial time algorithms.

The original motivation for this research was the production of terrain databases for training simulators or geographical information systems. This work is likely to be useful for machine vision and autonomous vehicle guidance applications as well.

Acknowledgements

This work was supported by a grant from the Florida High Technology and Industry Council.

REFERENCES


Estimating Motion Parameters in Linear Conformal Motion

Sanjoy K. Mishra and Dmitry B. Goldgof
Department of Computer Science and Engineering
University of South Florida
Tampa, Florida 33620

Abstract
This paper describes a curvature-based approach to nonrigid motion analysis. Based on surface curvature, it is possible to classify nonrigid motion into several classes: rigid, isometric, conformal, and general. In this work, we investigate local surface changes during conformal motion. We present the algorithm for local stretching recovery from Gaussian curvature based on linear approximations of the stretching function.

2 Classification of motion
There exist a rigorous theory of surfaces which uses differential geometry [Gra35, O'N66, Lip69] and considers various mappings between two surfaces. It has been established that 3-D smooth surfaces are uniquely characterized by first and second fundamental forms. Further, it has been shown that the Gaussian curvature is independent of object motion in space and parametrization [Gra35]. Thus, changes in Gaussian curvature gives nature of local motion in the object space.

We can classify the general motions (or transformations) of objects into several classes. We will use mean($H$) and Gaussian($K$) curvatures at each point before and after motion for our classification. We can define the following classes of motion: rigid, isometric, homothetic, conformal and general (non-conformal).

Rigid body motion is generally defined as a motion with no bendings. In terms of $H$ and $K$ it can be specified as a motion which preserves both $H$ and $K$ at all points. Isometric motion is a mapping which preserves distances along the surfaces as well as angles between curves on the surface. It is a motion which preserves Gaussian but not the mean curvature.

Conformal motion can be defined as a motion which preserves angles between curves on the surface but not the distances. More explicitly, the mapping (motion) of $S$ to $S'$ is conformal if the angle between two directed curves through a point $P$ of $S$ is equal to the angle between the two corresponding directed curves through the corresponding point $P'$ of $S'$ [Gra35].
3 Curvature Changes During Conformal Motion

Motion analysis for rigid body requires only the recovery of the translation vector and the rotation matrix. However, for the nonrigid motion we also need to consider local deformations such as surface stretching. In the rest of the paper, we describe a method to estimate local surface stretching for nonrigid motion with known point correspondences.

Let \( S \) and \( S' \) are surfaces before and after a motion. Then, a necessary and sufficient condition for the motion to be conformal is that the linear elements of \( S \) and \( S' \) be proportional:

\[
E'du^2 + 2F'dudu + G'dv^2 = t^2(Edu^2 + 2Fdudu + Gdv^2)
\]

or

\[
\frac{E'}{E} = \frac{F'}{F} = \frac{G'}{G} = t^2
\]  

(1)

In equation (2) \( t \) is the ratio of coefficient of the first fundamental form, is also infinitesimal stretching parameter for a point on the surface. A conformal motion is called homothetic if \( t \) is constant for all points on surface. It is easy to see that isometric motion is a conformal motion in which \( t = 1 \) at all points on surface. Moreover, the Gaussian curvature is a function of \( E, F \) and \( G \) only. It has been derived [GLH88] that under homothetic assumption

\[
K' = \frac{K}{t^2}
\]  

(2)

Equation (2) tells us that it is characteristic of a conformal motion that corresponding infinitesimal distances at corresponding points are proportional. The factor of proportionality, \( t \), depends only on the pair of corresponding points chosen. Thus from above discussion, we can derive conformal motion from homothetic motion (3) and a deviation function \( f(E,F,G,t) \).

\[
K' = \frac{K}{t^2} + f(E,F,G,t)
\]  

(3)

In rest of this section we will show that it possible to estimate the stretching function \( f \) with certain approximation of \( t \). We know that

\[
K = \frac{ln - m^2}{EG - F^2}
\]  

(4)

where \( l, m, \) and \( n \) coefficient of second order. Also we know that

\[
l - m^2 = F[(\Gamma_{12})_u - (\Gamma_{12})_v] + (\Gamma_{13})_u + (\Gamma_{23})_v - (\Gamma_{21})_u - (\Gamma_{12})_v + (\Gamma_{13})_u + (\Gamma_{23})_v
\]  

and

\[
\Gamma_{ij} = \frac{GE_u - 2FF_u + FE_u}{2(EG - F^2)}
\]  

(5)

where \( \Gamma_{ij} \) are Christoffel symbols of second kind which can be expressed in terms of \( E, F, G \) and their derivatives.

\[
\Gamma_{11} = \frac{GE_u - 2FF_u + FE_u}{2(EG - F^2)}
\]

\[
\Gamma_{12} = \frac{2GE_u - GG_u - FG_u}{2(EG - F^2)}
\]

\[
\Gamma_{13} = \frac{2GF_u - EE_u - FE_u}{2(EG - F^2)}
\]

\[
\Gamma_{21} = \frac{EG_u - FF_u - GE_u}{2(EG - F^2)}
\]

\[
\Gamma_{22} = \frac{2GE_u - GF_u + GG_u}{2(EG - F^2)}
\]  

(6)

Now from equation (2) we have

\[
E' = t^2E \quad F' = t^2F \quad G' = t^2G
\]  

(7)

Taking first derivative with respect to \( u \) and \( v \) we have

\[
E'_u = 2tt_uE + t^2E_u \quad F'_u = 2tt_uF + t^2F_u
\]

\[
G'_u = 2tt_uG + t^2G_u \quad E'_v = 2tt_vE + t^2E_v
\]

\[
F'_v = 2tt_vF + t^2F_v \quad G'_v = 2tt_vG + t^2G_v
\]  

(8)

The function \( f \) in equation (4) shows deviation of homothetic motion from conformal motion. If \( f \) is neglected the motion is homothetic. In the rest of the paper we derive the local stretching function \( f \) and show how parameters \( a, b, \) and \( c \) can be estimated. In particular, we consider linear approximation of the stretching function \( f(u,v) \). We will call this type of conformal motion linear conformal motion.

\[
t(u,v) = au + bv + c
\]  

(9)

Substituting \( t \) from equation (10) in (4) we obtain the expression for \( K' \) (Gaussian curvature after the motion).

\[
K' = \frac{K}{(au + bv + c)} + f(E,F,G,a,b,c)
\]  

(10)

In order to derive equation (11), we compute derivative of Christoffel symbols in terms of constants of first fundamental form. We substitute (8) and (9) in equations (7) to obtain values of Christoffel symbol after motion. We get following result.

\[
\Gamma_{11}' = \Gamma_{11} + f_{11} \quad \Gamma_{12}' = \Gamma_{12} + f_{12}
\]

\[
\Gamma_{21}' = \Gamma_{21} + f_{21} \quad \Gamma_{22}' = \Gamma_{22} + f_{22}
\]

where

\[
f_{11} = \frac{EG_{uu} - 2F^2_{uu} + FE_{uu}}{(EG - F^2)^2}
\]

\[
f_{12} = \frac{EG_{uv} - FG_{uv}}{(EG - F^2)^2}
\]

\[
f_{21} = \frac{2GF_{uu} - GG_{uu} - FG_{uu}}{(EG - F^2)^2}
\]

\[
f_{22} = \frac{2EF_{uu} - EE_{uu} + FE_{uu}}{(EG - F^2)^2}
\]

(11)
\[ f_{1z}^2 = \frac{E G_{v} - 2 F E_{v}}{(E G - F_{v})^2} \]
\[ f_{2z}^2 = \frac{E G_{u} - 2 F E_{u}}{(E G - F_{u})^2} \]

Also the partial derivatives of \( f_{i}^2 \) w.r.t \( u \) and \( v \) are required to be computed. Substituting these values in equation \( (5) \) we get equation of the form \( (11) \).

4 Estimating Surface Stretching in Linear Conformal Motion

It turns out that equation \( (11) \) is quadratic in three unknowns namely \( a, b \) and \( c \). Under basic assumption of correspondences between certain points on surface before and after motion, we can get a system of three nonlinear equations considering three different points.

It should be noted that point correspondences are given in different coordinate systems. Hence, \( E \) and \( E' \) can’t be computed directly since we don’t know the correspondence between coordinate directions. We can compute \( K \) and \( K' \) since Gaussian curvature does not depend on coordinate system. It is also possible to compute \( E'' \) by \( F \) \( G \) because we only need those coefficients at one (after the motion) time instant.

The system of polynomial equations of the form \( (11) \) can be written as \( C(A) = 0 \) or \( C_i(a, b, c) = 0 \) \( i = 1, 2 \) \( 3 \) where \( A \) is a vector of \( a, b, c \). This system of equation can be solved by at least two methods. In first method we reduce the three polynomial equations in three unknowns into a single polynomial with single unknown. This polynomial equation can be solved using any iterative methods [Ric86]. Other two unknowns are computed similarly using back substitution. This method is sufficient when we have small number of unknowns as in linear approximation of \( t \).

In second method we use homotopy algorithm [Mor87, JI88]. Solution using this method requires a basis function \( B_i(a, b, c) \). The \( B_i \) are chosen this way because their solutions are known and they have same degree as \( C_i \). Now a new function using \( B_i \) and \( C_i \) can be written as

\[ D(A, d) = (1 - d) B(A) + d C(A) \]

When \( d = 0 \), \( D(A, d) = B(A) \), and when \( d = 1 \), \( D(A, d) = C(A) \), which is the system we need to solve. Under some regularity conditions, the solution of \( D(A, d) = 0 \) trace path from trivial solutions of \( B(A) = 0 \) to desired solution of \( C(A) = 0 \) as \( t \) trace from 0 to 1. In order to follow the path, \( D(A + \Delta A, d) = 0 \) is solved for \( \Delta A \) for values of \( d \) ranging from 0 to 1. In order to solve this we use following approximation

\[ D(A_0 + \Delta A, d_0 + \Delta d) \approx D(A_0, d_0 + \Delta d) + \frac{\partial D}{\partial A} \Delta A \]

where

\[ \Delta A \approx \frac{D(A_0, d_0 + \Delta d)}{\frac{\partial D}{\partial A}} \]

This procedure breaks down if \( \frac{\partial D}{\partial A} \equiv 0 \).

The solution gives us initial approximation of \( \Delta A \). More exact determinations to \( \Delta A \) are made via Newton’s method [Bur81, Ric86] of other approximation schemes. Once \( \Delta A \) is established to requested accuracy, \( d \) is incremented and process is repeated up to \( d = 1 \). Alternatively, one can use a different set of polynomials if it is known that they have same solution set.

Thus we can estimate local motion parameter given a set of point correspondence. We require at least three point correspondence to solve for three unknowns. However, due to noise in data sets we may use more point correspondences then the actual minimum requirement.

5 Discussion and Future Research

In this paper we considered conformal motion of surfaces. In particular, we have shown how to extract local motion parameters (local surface stretching) given the surface before and after the motion and point correspondences. Further research include the extension of curvature-based algorithms to more general motions, application of the algorithms to simulated and real data, and investigation of noise sensitivity of the proposed algorithms.

This work is a part of ongoing research in the new area of nonrigid motion analysis. Applications of the derived algorithms include the motion analysis of the heart and, in particular, the left ventricle, where the local stretching factor provides the direct information on the condition of the heart muscle. Other applications of these methods include not only biomedical problems (such as study of heart and lungs), but also extend to such diverse areas as human face recognition for high-speed, low-bandwidth teleconferencing and material deformation studies.

6 Acknowledgement

This work is supported in part by NIH Grant 2108-085-L0 and in part by USF/DSR Research and Creative Scholarship Grant 2108-931-R0.

References

Weber and Schmidt, Boston, 1981.


* But you would not be wrong to suspect a page count fault.
Towards a Representation of Facial Expressions
for Recognition and Display

Chandra Karnbhamettu
Dmitry B. Goldgof
Alade Tokuta

Department of Computer Science and Engineering
University of South Florida
Tampa, Florida 33620 U.S.A.

Abstract

Development of a convenient model suitable for both simulation of facial expression and their recognition is suggested.

Our approach is to propose hybrid representation for facial expressions with a small number of parameters involved. The parameters are recovered using surface fitting technique. The model can be used for both simulation and recognition. The face is divided into patches, defined by fixed and movable nodal points. Movement of the movable nodal points on expression generation are recovered for each patch by surface fitting. The model is tested for a cubic patch.

We conclude with a discussion on extension of the model, wherein higher-order patches may be used to actually simulate the facial structure.

1 INTRODUCTION

The problem of modeling the facial expression involves two levels: action-based notation and structure-based representation. The first level deals with the problem of detecting the various muscle movements for different expressions on the face. There are various notational systems already developed in this area. The main goal of such systems include completeness, extensibility, and adaptability.

Sutton notation system [Val78] is one of these, which is more pictorially oriented. It records body position, including facial expression. Another scheme, by Birdwhistell [Bir70] was designed to record the detailed motion and action of body language and movement in interpersonal communication. However this lacks generalization of interaction between facial gestures.

The prominent among all these action based representations is facial action coding system, FACS [PW77]. It describes the set of all possible basic human actions performable on a human face called 'Action units'. It defines various basic actions, also called AU's, which interact in different fashion to build up a complete facial expression. Sample AU's may well be Cheek raiser, Brow raiser, Lip tightener etc. The AU's are closely connected to the anatomy of the human face. Each AU represents the movements of muscles involved in the expression.

For example, the Frontalis muscle is involved in raising of inner brow. Hence the AU, inner brow raiser takes into consideration the contraction and relaxation of this muscle. FACS provides all the possible mnemonic data required for various expressions. This mnemonic symbol is very helpful in terms of computer use. This theory may well be extended to other deformable structures like heart.

At the next level, structure-based representation involves the representation of the internal structure of the human face. In this level, the action representation is interpreted and the simulation is performed. The quality of any images produced will rest on this representation. There are various models for this representation which vary in the complexity of the techniques used to perform the facial actions.

The 2D surface patch technique [Hac77] involves warping of skin patches and rotating, translating each skin patch depending on the expression to be generated. The skin patches are conveniently identified and selected for the structural representation.

Another way of representation is by low-level simulation of the face [M81]. Skin is considered as a 2D surface, covering entire 3D structure, defined by various 3D points interconnected by arcs. The facial structure is considered as having three layers. Innermost layer is the bone structure and the outermost layer is the skin surface. Between these two layers is the muscle structure, defined by group of points, representing muscle fibers with an arc stretching from the fiber point to one or more skin points. Arcs are the media to store and transfer information regarding muscle movements.

Parke’s parametric model [IS2] is an impressive attempt to parametrize the facial structure. This 3-dimensional parametric model produces shaded facial images. The face, constructed of polygonal surfaces, is manipulated through the use of parameters which control interpolation, translation, rotation and scaling of the various facial features.

With this model, very little input information is needed to specify and generate a specific expression. The model has been successfully used to produce a large variety of facial image and several animated sequences. Face is classified by a set of parameters defining size attributes of the facial subsections.

Proceedings of the 3rd Florida Artificial Intelligence Research Symposium, 1990, by the Florida AI Research Society

c. 1990 FLAIRS 0-9620-1732-9/90/0350-0085

85
Expressions were coded as variations of these parameters, and animations were performed by the interpolation of change in expression. Obviously, as the number of parameters increase, the subtlety in facial expression increases.

The problem of representing the internal structure of the human face is not entirely solved. I.e., there is no efficient model to date which would represent the face in a flexible manner. There are still issues concerning which model is more flexible and efficient.

The objective of any model is to reduce the number of parameters used in the computation and yet achieve a flexibility in the final representation. Until now there is no model which could represent all the possibilities of a facial expression, like creasing and buckling of muscles, which produce and represent some important movements on the face [Wat57]. In addition, the uniform flow of skin over the muscles is difficult to achieve.

A more fundamental approach to the construction of the face needs to be taken that could encapsulate the widest range of facial types.

Face recognition by parametrization is mainly aimed at detecting the movement of various facial subsections, using the surface characteristics of the skin rather than concerning about motivators of the dynamics involved. The motivators of the dynamic characteristics are complex in modeling complex actions such as lower face jaw rotations. It is obvious that the skin, being supported by bone and multiple layers of muscle, produces thousands of movements. So it is not required to exactly simulate the neurons, muscles and joints, but a model with a minimum number of parameters that could emulate the primary features is required. These parameters are not aimed to model the biochemical or neuropsychological mechanisms taking place in skin movement. Their goal is to simulate the movement of the nodal points during an expression. So the recognition mainly involves selection of suitable and minimum number of parameters which can define the skin movement optimally. The recognition problem is simplified by dividing skin into various subsections, or patches. The basic problem involved in this representation is in the selection of skin patches. The skin patches have to be chosen so as to cover the entire combinations of muscle movements to generate different facial expressions. The selection of patches depends on the complexity of the particular facial subsection under consideration. The number of patches however has to be minimal, so as to reduce the information bandwidth and to minimize the problem of recognition of patch edges. The type of patch used to define the skin patch is also a point of discussion.

2 FACE MODELING BY SURFACE FIT

Surface fitting technique may be used to track the movements of each movable node in expression generation. The same model can be used for both the simulation and recognition of facial expression. Our approach to the problem involves reducing the number of regions to be considered on the face by defining patches to encapsulate most of them. Patches may be chosen so as to include various different sections of face. These individual patches may be warped to get the resulting facial structure. However, it is difficult to extract the necessary patches from the facial structure.

The basic problem involved here being the selection of patch edges from the face. It is desired to have a minimum number of patches to deal with and attain maximum flexibility in skin movements. Hence we are interested in choosing more patches in places where there is a maximum movement. For example, the portion around mouth is potential to have larger number of patches when compared to other places like those on the cheek.

Each skin-patch on the face is well-defined by fixed and movable nodes. An expression on face involves deformation of corresponding patches on face, which is due to the change of position of corresponding movable nodal points. Our approach to recognition involves using surface fitting techniques for the recognition of muscle movements of the various expressions. The surface fitting of the patch determines the surface parameters of the patch under consideration. These parameters contain the actual movement of the patch, which is involved in expression generation. Hence, the total number of these 'motion parameters' in recognition may be reduced by conveniently choosing the patch edges on face. The minimum number of patches defining face may well reduce the total number of parameters in recognition of expression.

Thus, the muscle movements are detected by fitting the corresponding patch and finding out the surface parameters after the movement. These parameters, along with information about the arrangement of movable nodes on each patch derivate the motion of the movable node.

We can determine the corresponding muscle movements of the patch through the information generated about the motion of the movable node. The motion parameters generated in the nodal point recovery process may be sent to the reconstruction step. Reconstruction of patch deformation is possible using these parameters. The movement of each nodal point involved in motion may be recovered on the model at recognition step.

3 BASIC STRUCTURE

The skin-patch is considered to have two types of nodal points, fixed and movable. The selection of fixed and movable nodes may be done depending on the region they belong to. For example, movable points on the cheek may be selected, since cheek plays a very important role in human expression. This movable point may be viewed as being controlled by the muscle Orbicularis oris, which controls the movements of cheek. The displacement of movable nodal points can be used to detect muscle movements: both fixed and movable points define the patch. The fixed points define the basic structure of the face, and on the other hand movable points derive movements of each individual sections of the face. The structural organization of the proposed model includes the following three phases.

3.1 Face data acquisition.

This phase deals with the capturing of data points of face by using a 2-D or 3-D camera. We assume 3D data acquisition capability, which provides points on the face surface. It may involve a computer controlled camera, which captures the facial image at regular intervals. The camera processor would interpret these images into the movements of points on the face. The output of this phase is the values of x,y,z of each point for each instant.
3.2 Nodal point recovery (recognition).

This phase deals with the recovery of the movable nodal points. Each muscle movement causes the nodal points to change their position. This affects only the moving nodal points. The change in position of the movable points may be detected by the use of information generated by the data acquisition step. For example, consider a facial expression, cheek raiser to be simulated. The muscles involved in this are, Orbicularis oculi and pars orbitalis. Hence the movable points controlled by these muscles will be moved due to this expression. Also, the neighboring movable nodal points adjacent to these points will be moved due to this motion. These movements may be in any direction.

The change in position of the points generated by data acquisition step provides the information about possible movement of each patch under consideration. Hence this information may be used to fit the patch under consideration and determine the surface parameters of the patch. This process may be repeated for each patch on the face, which may generate a set of parameters, providing the necessary information of muscle movement. These parameters may be used to get the motion of movable nodes through the information about position of each movable node. Thus, the motion parameters of each movable node involved in the expression generation may be determined. The parameters are later encoded and sent to the next phase. The optimized version of this model is aimed at decreasing the number of skin patches, preferably a single skin patch corresponding to each expression generation.

A notation similar to the facial action coding system may be used in representing an expression in terms of skin patches. Basic action units may be devised based on the FACS notation. Surface fitting approximates the position of each skin patch involved in the expression. The warping of these skin patches along with all the other skin patches not involved in the expression generates the whole expression on the face.

The output of this phase may be viewed as the encoded form of muscle movements for each patch. The important issue here is the number of parameters being sent from this phase. It is desired to send minimum number of parameters from this phase to the reconstruction phase. The amount of information sent from this phase depend on the number of parameters involved in the whole process and also on the encoding of these parameters. The main goal being the reduction of information bandwidth in visual communications.

3.3 Face surface reconstruction.

This phase involves the reconstruction of the facial surface from the fixed and movable nodal points. It receives the 'encoded' form of the muscle movements through parameters generated during nodal point recovery process. This data generated by nodal point recovery step provides the information about the movements of different movable nodes due to an expression generation. Hence the parameters received by this phase are used to construct the deformed skin patch. It decodes these movements and passes over to the lower level graphical routines for the graphical display. Basically it involves reconstructing each skin patch under consideration using their corresponding surface data.

The encoded form of parameters of each surface patch is decoded, thus reconstructing the movement of each patch. These are then displayed using two kinds of information. One is the information generated by nodal point recovery step in evaluating the movement of movable nodes of each patch. Second, the information about fixed nodal points on each patch which is available from the face model. Hence, both fixed and movable nodal points are considered here for display.

The face model, or the neutral image defined by the fixed nodal points would be already available for this phase i.e., structure of the face prior to received movable nodal points can be reconstructed using 'expressionless' facial data. The information generated by nodal point recovery step provides information of the movable points affected by the expression. This information, along with the face model may be used to reconstruct the facial expression. Thus, we are able to find out the new position of the patch after muscle movement. This surface may be displayed using graphic routines for each time instance.

4 Implementation of a Simplified PATCH

The following cubic patch is used to demonstrate the suggested approach.

\[ z(x, y) = ax^2 + by^2 + cz^2 + dxy + eyz + fx + gy + h \]  

(1)

Changes in the ten parameters \(a, b, c, d, e, f, g, h, i, j\) correspond to the movement of the skin patch.

The points \((x, y)\) define the nodal points over the patch. Supposing that the above equation represents one patch over the face, using a surface fit operation we obtain the corresponding values of the parameters.

In our implementation, the above patch is defined by 21 total nodal points. The patch is observed by varying a movable nodal point and reconstructing the patch after motion. Figures 1 to 4 correspond to the various stages of the above process. Figure 1 shows the patch before the motion. The patch in figure 3 is obtained using the information generated by sampling the patch. The sampled data may be viewed as the data generated by camera. Fitting of this data gives the surface parameters of the patch, before motion. This is essentially the output of data acquisition step. These parameters are used to generate the patch in the reconstruction phase.

The patch is now subjected to deformation by moving one of the movable nodal points. Since the magnitude and direction of the change in shape is controlled by the nodal points around that area, the patch deformation is seen around the nodal point which is moved. The patch after motion reconstructed from the nodal points is shown in figure 2. Figure 4 represents the patch after motion reconstructed from sampled points. This is obtained due to the information generated by the camera, i.e., sampled points in our case. The sample points after motion is fitted to get the surface parameters of the deformed patch. These parameters are the output of data acquisition step after motion.
The patch in the figures is defined by 21 nodal points, out of which 1 is movable. The difference between figures 1 and 2 is because of the motion of one of the 21 movable nodal points, which is situated in the middle of the patch. Figure 3 is reconstructed from figure 1, using reconstruction step. The similarity of these two patches verifies the reconstruction. Figure 2 is patch after the motion of the movable nodal point. The motion of a movable node is assumed to be in vertical direction in the simplified case. The vertical elevation of patch around the center is because of motion of the movable nodal point. The patch in figure 4 is the one after reconstruction. Again the similarity in patches verifies the reconstruction of nodal movement.

Actual displacement of the movable node : 0.9
Recovered displacement of the movable node : 0.898907

The technique presented here has also been tested using a deformation model, which generates the position of movable nodes effected by the deformation specified.

Here, the movement of nodal points are not restricted to vertical direction only. Recognition is done using the intersection coordinates of muscle vector and the skin patch.

Following are the results of the second case.

Position of the movable node,
before deformation : (32.0, 32.0, 0.1)
Position of the movable node,
after deformation : (22.08, 22.08, 0.077644)
Recovered position of the movable node : (22.02, 22.02, 0.078098)

It may also be noted that the error is introduced due to the surface approximation of the patch from the given nodal points. Using more flexible surface patch model will reduce that error.

5 CONCLUSION

In this paper we present a new approach for modeling expressions of the human face. It suggests analytic surface representation suitable for both recognition of muscle movements and display of the face surfaces. Cubic patch was used to demonstrate and test the suggested algorithm. It is clear that this does not correspond entirely with the real skin patch. However, it is suitable for early experiments to demonstrate our approach. As the complexity in the skin patch equation increases, flexibility of movement of the patch increases. The above patch is easily extended to more complicated surface patches such as higher order polynomials or superquadric surfaces [H81, ED87, J88]. Future research include the investigation of the best choice for surface description as well as more realistic movable nodal points motion model.

6 ACKNOWLEDGEMENT.

This work is supported in part by FHTIC Grants 2108-984-L0 and 2108-985-L0.

References


ICONKAT: Integrated Constructivist Knowledge Acquisition Tool

H. Stahl\textsuperscript{1}, K. M. Ford\textsuperscript{2}, J. R. Adams-Webber\textsuperscript{3} and J. Novak\textsuperscript{1}

\textsuperscript{1}Cornell University
Ithaca, New York

\textsuperscript{2}The University of West Florida
Pensacola, Florida

\textsuperscript{3}Brock University
St. Catharines, Ontario, Canada

ABSTRACT

The purpose of this paper is to report on a continuing research effort aimed at the development of a new automated knowledge acquisition tool, ICONKAT. Established on the basis of both Personal Construct Theory (Kelly, 1955) and Assimilation Theory (Ausubel, 1963), ICONKAT assists in the elicitation of a domain expert's object space as well as the rules employed to reason over that structured space. Thus, ICONKAT directly supports knowledge acquisition for the recent hybrid AI performance environments (e.g., NEXPERT) that utilize the twin orthogonal dimensions of representation (i.e., objects and classes) and reasoning (i.e., rules). The Integrated Constructivist Knowledge Acquisition Tool (ICONKAT) discussed here is currently employed in the design and construction of a medical diagnostic expert system.

1. INTRODUCTION

Knowledge acquisition involves both the elicitation of human expertise and its subsequent constructive modeling in a computer system — thus permitting such expertise to be made widely available, and hence useful to society in general. The development of expert systems has typically involved the knowledge engineer's working closely with a specialist to acquire the relevant knowledge of the specialist's domain. This process has two significant drawbacks: it has been extremely laborious and domain experts often have difficulty articulating their knowledge in forms useful to the knowledge engineer. Indeed, Feigenbaum (1977) has suggested that knowledge acquisition is "the bottleneck problem of applied artificial intelligence."

A commonly proposed partial solution to the knowledge acquisition bottleneck is the design and implementation of automated tools for the purpose of interacting with domain experts, acquiring and organizing knowledge, and automatically generating a prototype expert system. However, as noted by Bradshaw (1989), a major difficulty has been that much of this work lacks a plausible theoretical foundation.

As a consequence of incomplete theory and a limited repertoire of practical approaches to the dynamics of the modeling process, knowledge engineers have had to rely on intuition and experience as the primary means of developing and testing effective procedures. (p. 6)

Rarely do working knowledge engineers explicitly consider the epistemological underpinnings of the methods and tools they employ in their task. Many of those engaged in knowledge acquisition (as researchers or practitioners) may be classified as tool-makers and/or tool-users. Tool-makers should exploit theory as a means of building their tools on a sound footing and as a framework in which to make explicit their epistemological assumptions (Ford & Adams-Webber, 1990). Furthermore, theory may offer tool-makers a useful infrastructure upon which to build highly integrated collections of tools and techniques (i.e., hybrid knowledge acquisition workbenches).

Integration is now the battle cry of tool-makers rushing to develop hybrid knowledge acquisition tools. However, it is becoming widely realized that the ad hoc combinations of techniques and tools — a sort of Occam's hell — may not contribute much to ameliorating the knowledge acquisition bottleneck. In contrast, we have employed theories of human cognitive processes to develop ICONKAT, an integrated hybrid knowledge acquisition tool. Our system is based on constructivist psychology and epistemology. It utilizes ideas and tools rooted in constructivism and integrated together in a highly principled manner.
2. PERSONAL CONSTRUCT THEORY

Psychology research has often been used to improve communication between knowledge engineers and domain experts (Bainbridge, 1979; Ericsson & Simon, 1984; Hoffman, 1987). In particular, Kelly's (1955) Personal Construct Theory has served as a basis for several approaches to the design and construction of computer-based knowledge acquisition tools (Boose, 1984; Boose & Bradshaw, 1987; Ford, Petry, Adams-Webber and Chang, 1990; Gaines & Shaw, 1986). Personal construct theory, as formulated by Kelly and summarized by Adams-Webber (1987), assumes that people typically use cognitive dimensions termed "constructs" to evaluate their experience. Each construct, by definition, represents a single bipolar distinction. For example, a mother might use the construct 'excited/calm', among others, to interpret her children's behavior. The underlying relation between the alternative poles of any construct is essentially one of bipolar opposition, or 'contrariety' (Husain, 1983).

Kelly (1955; p.68) postis also that "Each construct is convenient for the anticipation of a finite range of events only." It follows that each of a person's constructs has a specific range of convenience, which constitutes "all those things to which the user would find its application useful." Moreover, any particular construct may have a somewhat different 'context' for each person who uses it, where its context comprises "the elements lying within the range of convenience of the construct (Kelly, 1955; pp. 108-109)." The range of convenience of a construct also limits its logical extension, since outside of its range of convenience the construct is not applicable.

From a Kellyan perspective, a necessary condition for organized thought is some degree of overlap between constructs in terms of their respective ranges of convenience (Adams-Webber, 1970). It is this overlap (intersection) between the constructs' ranges of convenience that enables an event to be anticipated. His model implies that persons seek to predict and control events by forming relevant hypotheses, and then testing them against available evidence. These hypotheses are derived from the relationships between constructs that articulate the 'logical' structure of an individual's personal construct system. More precisely, Kelly's (1955; p. 56) organization corollary asserts that "Each person characteristically evolves for his own convenience in anticipating events, a construction system embracing ordinal relationships between constructs." Much of the research aimed at application of personal construct theory to the knowledge acquisition process is concerned primarily with the development of new methods for eliciting the relationships between an individual's constructs on the basis of repertory grid data.

Repertory grid analysis is becoming an increasingly useful tool in knowledge acquisition for expert systems; the root of the method's relative success at bridging the gulf between the semantic depth of the domain expert's knowledge and the limited nature of our entirely syntactic tools (i.e., computers) lies in the fact that the repertory grid is the major methodological component of personal construct theory -- a complete and epistemologically sophisticated psychology (Ford & Adams-Webber, 1990). The judicious use of repertory grids may assist the knowledge engineer in avoiding the domain expert's cognitive defenses and thus elicit deep knowledge that the expert would not have been able to express otherwise.

2.1 Repertory Grids

Kelly's Role Construct Repertory Grid, was developed to provide a window into an individual's personal construct system. Commonly referred to as the 'repertory grid', Kelly's tool 'expresses one's own finite system of cross references between the personal observations he has made and the personal constructs he has erected' (1965, p.291). The grid is essentially a complex sorting task in which a list of elements are judged successively against a set of bipolar constructs (Adams-Webber, 1987). The elements can be either concrete or abstract entities, and may be said to operationally define the grid's universe of discourse. These elements should be chosen carefully to completely represent the topic of interest and also be roughly of the same type and level of complexity. Constructs represent the ways in which elements are judged to be similar or different from each other, that is, they permit the subject to make relevant distinctions among the elements. Thus, a repertory grid may be considered a mapping of elements onto constructs. The data generated by each subject are entered into a separate two-dimensional table, or 'grid', in which there is a column for every element and a row for every construct. Each row-column intersect in this table contains a number indicating how a given construct was applied to a particular element.

2.1.2 Repertory Grid Tools and Techniques

During the last thirty years there has been a considerable proliferation of new forms of repertory grid tests in which people, objects, situations, or other kinds of elements, are either categorized, rated, or rank-ordered on a set of constructs. The act of the respondent's assigning a rating to an element on a given construct has been interpreted in a variety of ways, and has thus lead to several different approaches to grid analysis, including information theoretic measures, non-parametric factor analysis, conventional factor analysis, principal component analysis, multidimensional scaling, and hierarchical cluster analysis, among others (Adams-Webber, 1987). These programs have been widely used in clinical psychology, as well as in the study of education and management.
Although extremely useful in some contexts, these approaches share the limitation that they can portray only symmetric relations. In other words, although they are able to detect that a given construct is similar to another construct, such methods cannot derive one-sided inferential relations (i.e., they cannot show that one construct logically entails another). Paradoxically, this is a limitation of conventional distance-based, cluster-analytic approaches, some researchers have developed procedures for the logical analysis of construct entailment (Gaines & Shaw, 1980; Ford, 1987).

The ICONKAT system includes both a repertory grid elicitation and rule generation component, largely based on NICOD (Ford et. al, 1990), a semi-automated knowledge acquisition system which generates production rules from repertory grid data. The system finds logical entailments and substantiates generalizations between the constructs. The tool has previously been employed in the capture of radiological knowledge (mammography) that the domain experts (radiologists) were otherwise unable to articulate (Ford et. al, 1990).

3. ASSIMILATION THEORY

Assimilation theory, like personal construct psychology theory, belongs to the family of theories developing a constructivist model of human representational processes. Developed by David Ausubel, this cognitive theory of learning and education explicitly rejects the views of behavioral psychologists and argues that learning is synonymous with a change in the meanings of one’s experiences. Ausubel’s fundamental premise seems deceptively simple:

*Meaningful learning results when new information is acquired by deliberate effort on the part of the learner to link the new information with relevant, preexisting concepts or propositions in the learner’s own cognitive structure.* (1978, p. 159)

Ausubel refers to the process of linking new information to existing segments of cognitive structure as subsumption, and his theory examines various forms of meaningful (as opposed to rote) learning that involve the assimilation of new information. Meaningful learning requires that the learner’s cognitive framework contain relevant anchoring ideas to which the new material can be related. In fact, Ausubel has stated that “the most important single factor influencing learning is what the learner already knows. Ascertain this and teach him accordingly.”

From birth onward, each human being constructs his/her own concepts and propositions that create his/her views on how the universe works. We define concepts as a perceived regularity in events or objects, or records of events or objects, designated by a word or symbol (Novak & Gowin, 1984). Propositions are statements about the universe that combine two or more concepts and linking symbols; thus sky is blue and 2+2=4 are propositions composed of concepts. Constructivist psychology holds that each person constructs his/her own meaning based on the discovery of perceived regularities or by reception learning’ (Ausubel, 1963, 1978) where new concept meanings are acquired by learning new relationships between new concepts and concepts already known. Constructivist epistemology maintains that the creation of new knowledge extends one’s meaning making by contributing to new concepts (or new meanings to old concepts) and new propositions (such as $E=mc^2$). In sharp contrast to positivist epistemology, constructivist epistemology sees human meaning making as a continuous, evolutionary process (Toulmin, 1972). New knowledge construction usually requires the creation of new events or objects and/or new ways to make records of events or objects, and/or new ways to organize or transform these records. Thus, both the creation (an epistemological process) and learning (a psychological process) of knowledge depend upon innate human capacities for meaning making (Novak, 1977). Concept maps serve as a tool to facilitate the representation of knowledge structures, as well as the sharing of meanings.

3.1 Concept Maps

Concept maps are a two-dimensional representation of a set of concepts constructed so that the interrelationships among the concepts are evident. The vertical axis in a map expresses a hierarchical framework for the included concepts. More general, inclusive concepts are found at the highest levels of a map with progressively more specific, less inclusive concepts arranged below them. Maps emphasize the most general concepts by linking supporting ideas to them with propositions. Along the vertical axis, maps display Ausubel’s concept of subsumption, namely that new information often is relatable to and subsumable under more inclusive concepts. Along the horizontal axis, maps express the idea of progressive differentiation by showing how concepts gain greater meaning as new propositions are acquired. This axis in a map shows that meaning making is a continuous process. Concepts are always being modified and made more explicit and more inclusive.
The overall structure of a concept map creates a hierarchical framework for the included concepts. All concepts at a given level in the hierarchy will tend to have a similar degree of generality. The structure of a concept map depends entirely on context. Consequently, maps having similar concepts can vary from context to context. The strength of concept mapping lies in its ability to measure a particular person's knowledge structure about a given topic in a given context (Novak & Gowin, 1984). Research has shown that maps produced by experts and novices differ substantially, since expert practitioners develop their knowledge structure in novel and unique ways which do not necessarily coincide with widely shared consensual beliefs (Novak, 1990).

Concept maps represent meaningful relationships between concepts in the form of propositions. In the simplest form, a concept map would be just two concepts connected by a linking word to form a single proposition. For example, "grass is green" would represent a simple map forming a valid proposition about the concepts "grass" and "green". A concept acquires additional meanings as more propositions begin to include the concept. Thus, "grass is green", "grass is a plant", "grass grows" and so on all increase the meaning of the concept "grass". In this sense, concept maps represent concept meaning in a hierarchical framework of embedded propositions. Consider the concept map about concept mapping in Figure 1 (by convention links run top-down unless annotated with an arrowhead).

In an educational setting, concept mapping techniques have aided people of every age examine many fields of knowledge (Novak and Gowin, 1984; Novak, 1990). This rich expressive power derives from a map's ability to allow its creator the use of a virtually unlimited set of linking words to show how meanings have been construed. When concepts and linking words are carefully chosen, concept maps are powerful tools for observing nuances of meaning. Mapping techniques have also been employed to help students "learn how to learn" by bringing to the surface one's cognitive structure and self-constructed knowledge. It is exactly these problems that currently face the knowledge acquisition community. As a knowledge elicitation tool, concept maps provide a framework for eliciting the most significant part of a specialist's expertise, his/her personally constructed knowledge (Ford & Adams-Webber, 1990).

4. ICONKAT: An Integrated Constructivist Knowledge Acquisition Tool

A current trend in applied AI (e.g., expert system design) is the development of integrated, embeddable performance environments which offer knowledge engineers both object-oriented knowledge bases and rule-structured inferencing mechanisms. However, this development now requires that automated knowledge acquisition tools generate much more than merely
production rules when creating prototype systems. The constructivist tool described here (i.e., ICONKAT) interactively elicits the domain expert's model of the relevant object space by use of concept maps, as well as generating the associated rules from repertory grid data. An overview of the entire system is illustrated below in Figure 2.

ICONKAT interacts with an expert through a concept mapping and repertory grid interface. The system uses concept maps to elicit the object space from the domain expert, incrementally generating a hierarchically-structured knowledge base as a map is created. The knowledge base created from the map is provided to the representation dimension of the performance environment. Concepts in the map are specified by the user as either classes, objects, subobjects, or properties. The links are used to propagate inheritance appropriately. The concept mapping portion of ICONKAT also includes an advisory expert system (ExpertMAP), which assists the expert in creating well-structured maps. By monitoring the structure of the concept map under construction, advice and information are given about actions the user might reasonably take to develop and improve the map.

The system elicits the reasoning dimension by the use of repertory grids. The hierarchically-grouped concepts found in the elicited concept map provide the elements for a single repertory grid. A hierarchical collection of grids can, if necessary, be elicited on the basis of the object space, with each grid at a different level of abstraction. Using TRIADS and other well-known repertory grid techniques, constructs are elicited on the basis of the elements already defined as objects in the knowledge base. The constructs can then serve as the properties of these objects. Further analysis of the repertory grids will find the construct entailments necessary to generate production rules. Thus, the system creates a reasoning space structured on the basis of the object space already elicited and represented.

**SUMMARY**

Like experts in many other fields, knowledge engineers typically rely on intuition and personal experience, and rarely examine the theoretical implications of the assumptions underlying various methods and tools which they employ. In contrast, we have suggested that an explicit model of human cognitive processes, including perception, memory, representation, anticipation and reasoning, might serve to help us simplify some of the central problems in this field. In particular, we have discussed a new knowledge acquisition tool (ICONKAT) explicitly constructed to exploit the sound theoretical footing afforded by constructivist epistemology as elaborated in Personal Construct Theory and Assimilation Theory.

![Figure 2: ICONKAT System Architecture](image-url)
The research described here has been directed toward the development and unification of the underlying theoretical foundations for an integrated constructivist approach to knowledge acquisition. Specifically, two constructivist methodological tools, the concept map and repertory grid, have been briefly presented. Integrated in a principled manner, the resulting human-machine system (of which the user is an integral part) aims to bring the expert's self-constructed knowledge to the surface. The tool discussed here is currently employed in the design and construction of an expert system for the diagnosis of first pass cardiac functional images, a non-invasive radionuclide technique used to evaluate heart wall motion abnormalities.

ACKNOWLEDGEMENTS

Many people have influenced the development of this work. We owe particular thanks to John Brennan, John Coffey, John L. Liseth, Joe Novak and Jeff Yerkes. This work was supported in part by Florida High Technology Enhancement Funds.

REFERENCES


An Object-Oriented Knowledge Acquisition Tool*

by

PRATIK K. BISWAS**

Department of Computer Science
and
The Institute for Expert Systems and Robotics
The Florida State University
Tallahassee, Florida 32306-4019
U.S.A.

Keywords:

Expert systems, Knowledge acquisition, Knowledge-editor, Knowledge-base, Objects, Attributes, Value-classes, Association, Aggregation.

Abstract

The task of knowledge acquisition for expert systems is often expensive, time-consuming and inefficient. This problem can be best addressed by automating the whole process at its most fundamental level. This paper is on a research effort that has been aimed at this direction. A knowledge acquisition support tool has been developed for this purpose. This tool, better described as an 'intelligent knowledge-editor', enables the domain expert to build a knowledge-base by creating new rules from the attributes of the objects under consideration and their value-classes (domains). It also allows him to edit an existing knowledge-base by modifying, deleting, copying or moving old rules. An object-oriented approach has been used in its design.

Introduction

Expert systems have generated tremendous interest in recent years and have become applicable in different areas. But one of their obvious limitations has remained in the area of knowledge acquisition, a field concerned with eliciting knowledge from the domain expert.

Knowledge acquisition for expert systems typically involves prolonged interaction between the knowledge engineer and domain expert, whereby the expert's knowledge is incorporated into the system. This complex interplay can be avoided if the domain expert is provided with an editor which can transform his knowledge into an internally usable form, that can be subsequently modified. This may qualify as 'learning from instruction' which is defined as conversion of domain knowledge into an executable form. This paper describes the development of one such tool that assists the domain expert in coding his knowledge into an internal representation. However the accuracy and validity of such knowledge depend on the expert's ability, experience, correctness and consistency.

Object-Oriented Approach

An object-oriented data model, under Intellicorp's Knowledge Engineering Environment (KEE) on Symbolics-LISP-machine has been chosen in designing the system for such desirable features as data abstraction and program modularity. Each component of the system represents an object. Each object is characterized by its attributes. An object may be associated with another or can be aggregated with others into one.

---

* This research was undertaken at the Krupp Forschungsinstitut (KFI) in Essen, West Germany.

** On a visit to KFI from the Florida State University.
Two types of knowledge can be acquired. The first type, known as the declarative knowledge, consists of data and facts related to the problem domain. The second type, called the procedural knowledge, consists of rules for solving the problem. Our system allows the user to build and modify the procedural knowledge from the existing database.

The learning program, named as the "knowledge-acquisition-module" (KAM), builds up a learning system that allows the user to work on the knowledge database and the rule-class of his choice. He can either operate on an existing rule or create a new rule here.

If the user decides to create a new rule then he is prompted for the rule-type (Production or Deduction), rule-component (Premise or Conclusion) and the logical connectives. The user can create rules comprising of simple or compound clauses. The clause is built up from the attributes (slots) of the objects (units) and their values-class (domain), which are displayed as menu items. The system also displays a clause, premise, conclusion, external form (user friendly external representation) and the internal form (KEE representation) of a rule, just as they are completed.

If, however, the user decides to operate on an existing rule then he selects the particular rule from the rule-class and the rule operation (edit option). He can delete the chosen rule, copy it under a new name, move it to a new location (knowledge-base) or modify it. If he decides to enter the modifier then the external form of the rule is displayed on the editor window. A compound clause can be modified by changing the logical connectives, adding a new clause or by rebuilding the clause all over. A simple clause can be replaced by the newly built (modified) clause. The internal form of the modified rule then gets stored in the knowledge-base.

The translation (during rule modification and creation) from the external to the internal form of a rule and vice versa is done through the data-dictionary of the chosen knowledge-base.

The user can edit several rules of the selected rule-class or quit the knowledge-editor. He can only operate on one knowledge-base and one rule-class during one go. He can save the knowledge-base while exiting. If he wishes to work on a different knowledge-base or on a different rule-class then he has the option to restart the process.

Fig. 1 shows the configuration of the learning system.

**System Design**

The learning program (KAM), on being triggered, creates an object-oriented dynamic knowledge-base, named as knowledge-acquisition-system (KAS). KAS interacts with the expert and gathers his expertise, used subsequently to modify the knowledge-base. Conceptually the whole process can be visualized by the following object model (fig. 2).

The knowledge of KAS is stored inside two objects, namely screen-io and knowledge-editor. The object (unit) named screen-io contains slots for knowledge-bases, rule-classes-of-a-chosen-kb, operate-existing-rule, rules-of-a-chosen-class, rule-operation, etc. Active-images attached to these slots provide user friendly interface to elicit responses from the user which are then passed on to the other unit named knowledge-editor. This object (unit) contains slots to store menus for rule-types, rule-components, logical-constructs, property-lists, value-lists, logical-operators, numeric-operators, compound-clause-modification, clause-modification, etc. which are displayed during rule creation or modification. The user's selection of items from these various menus, enables the editor to construct the new or modified rule, thereby updating the knowledge-base. The complete object hierarchy of KAS is shown in fig. 3.

The translation from the external to internal form of a rule and vice versa is done by the KAM via an unit named data-dictionary of the chosen knowledge-base. This unit can be created programmatically by the KAM. It contains a slot named list-of-properties that has as its domain a list of the attributes of all the objects (units) under consideration. Each item in the list is an ordered pair of an external and an internal name of the slot.
Functionally the code for KAM has been divided into seven broad regions to enhance program modularity and user readability. These are as follows:

1. Functions that set up the system and create all units and co-ordinate knowledge editing.
2. Functions that control window displays during rule selection and creation.
3. Functions that create slots to store display menus, numeric inputs in the unit named knowledge-editor.
4. Functions that help to create a new rule.
5. Functions that provide user interface during rule modification.
6. Functions that help modify an existing rule.
7. Functions that translate the external form of the rule into a KEE representation and vice versa.

Is the model object-oriented?

In this section, we attempt to answer this question by looking into it from various angles.

We have used data abstractions consistent with object-oriented systems. Our system involves associations between various object types (fig. 2). Besides, the KAS is an aggregation of two object classes, which themselves are aggregations of several attributes (fig. 3).

Secondly the object classes of KAS are organized in a functional hierarchy, where each class possesses features like encapsulation and message-passing capabilities. These object classes disallow operations incompatible with the value-classes of their slots. User responses to the interfaces attached to the slots of these generic objects instantiate their types.

Lastly but not the least, the system has been implemented in an object-oriented environment. Though this does not necessarily make our system object-oriented, but atleast by identifying our objects with KEE objects, we could use KEE’s features of encapsulation, message-passing mechanism, structural and operational object-orientedness in our model.

Conclusion and Future Research

This paper focuses on a knowledge acquisition tool that replaces the expert to knowledge engineer communication with another, namely expert to program. This tool allows the expert to build and maintain a rule-base using the existing database. It extracts the knowledge from the expert by interacting with him through an user friendly interface. This interface is mostly menu-driven and uses the active-image facilities from the KEE programming environment. The acquired knowledge, stored temporarily in an object-oriented, dynamic knowledge-base (KAS) is used to update the knowledge-base of the expert system under review.

However it is only a computer aid to the domain expert and by no means full-proof. Using it facilitates the task of entering knowledge into the system and decreases the chance of typographic and syntactic errors. But it does not guarantee semantic consistency. As mentioned before, it is assumed that the domain expert has sufficient experience and knows what he is doing or that the knowledge engineer is intimately familiar with what he is maintaining. The idea behind the whole project was to build a tool to assist Krupp’s experts and
engineers in building and maintaining knowledge-bases within reasonable time and with bare minimum of effort. It was developed to supplement the expert system shell KEKE.

As for further research, the learner can be extended to do semantic checking to test for the validity, consistency and completeness of the acquired expertise. That will greatly enhance its ability to learn by instruction and remove this persistent bottleneck in the construction of expert systems.

Acknowledgements

This paper is based on research that was jointly supported by the Gesellschaft für Mathematik und Datenverarbeitung mbH (GMD) at Bonn, in West Germany, the Krupp Forschungsinstitut (Krupp research institute) at Essen, in West Germany and the International Computer Science Institute (ICSI) at Berkeley, in United States. It was undertaken at the KFI.

The author is grateful to Dr. S. Harris & Dr. B. Feisensfeld of KFI for proposing and organizing this research and to Werner Franzen & Thomas Koch of the same institute for their suggestions and assistance.

References


KASH: A General Purpose Knowledge Acquisition Shell

Christopher R. Westphal
Institute for Defense Analyses
1801 North Beauregard Street
Alexandria, Virginia 22311

Duc Tran
4923 Novak Lane
Fairfax, Virginia 22030

Abstract

The field of expert systems has claimed numerous and successful applications ranging from simple tasks, such as monitoring a valve or a field race, to very complex tasks that may include process scheduling or design. However, to construct a system requires the developer to isolate a set of rules that will guide the decision making process. Rules are conventionally defined during a series of interviews between a domain expert and a knowledge engineer. The encoding and representation of the extracted domain knowledge has proven to be a difficult barrier to overcome. KASH (Knowledge Acquisition Shell) has been designed to address this problem by providing a knowledge engineer with a set of utilities for constructing knowledge acquisition sessions based on interviewing techniques. The information elicited from domain experts during the sessions is guided by a question dependency graph (QDG).

The QDG, defined by the knowledge engineer, is a series of control questions about the domain that are used to organize the cognitive processes of an expert. The content information supplied by the expert, in response to the questions, is represented in the form of a concept map. These maps can be constructed in a top-down or bottom-up manner by the QDG and used by KASH to generate the rules for a large class of expert system domains.

Introduction

Knowledge acquisition tools use a variety of techniques to elicit from a domain expert the essential information for building an expert system. The application domains of these tools can be aligned into two categories: analysis (e.g., diagnosis, identification, interpretation) and synthesis (e.g., scheduling, planning, design). Analysis is a top-down process providing an examination of a set of goals that are decomposed into simpler and more basic elements and relationships. For example, FIX (Roth, Pierce and Dalton, 1989) is a diagnostic system based on class attributes for representing fault detection and isolation. Synthesis is a bottom-up process supporting the composition or combination of given facts and observations that are used to construct a set of goals. For example, SALT (Marcus, 1988) is a tool based on a propose-and-revise method for designing elevator configurations.

The above classes of tools work well within their intended domains, but are not easily transferred to other applications. Consequently, the time and effort required for producing the systems must be reinvested for each instance. This will increase the costs to the customer and subsequently decrease reusability and portability of the tools across domain applications. To address the different categories (analysis or synthesis), a system must be able to work within the constraints set by each. Expert system shells currently provide the capacity to build top-down or bottom-up applications by supplying the developer with a finite collection of objects to represent the domains. The manner in which the objects are defined and related to other objects determines the degree of acceptability of the system. Therefore, if the focus of producing a knowledge acquisition system is placed on the structure, rather than on content, a more diverse selection of applications may be addressed.

One knowledge acquisition system providing a domain-specific structure, KLAMShell (Cochran, 1988), was developed to aid in the construction of knowledge bases for maintenance and troubleshooting. The shell elicits information via subgoal satisfaction in a depth first manner to retain a context focus on the knowledge structure. Each goal defined is decomposed (via "push" menus) into a series of subgoals. The bottom-most nodes are transformed (via "pop" menus) into actions, such as questions or instructions, to be used in the final system. This process continues until all goals have been specified. The system is, however, domain specific and the generality of its guidance is limited to only a small subset of the domain, thereby restricting its use.

Another system that may be classified as a general purpose knowledge acquisition shell is PROTEGE (Musen, 1989). It is a tool capable of generating other knowledge acquisition systems using planning entities, task level actions, and input data. The methodologies used by PROTEGE separate the modeling of a domain (i.e., control structure) from the application knowledge (Musen, 1988), thereby customizing each system. However PROTEGE relies on a set of fixed templates to elicit information and this will limit the range of systems that can be produced.

From this perspective, KASH (Knowledge Acquisition Shell) has been defined as a domain-independent, knowledge elicitation shell. The shell provides a general purpose (reusable) environment for encoding the problem-solving methods of domain experts. A set of three independent modules (Figure 1) operate within the top-down (analysis) and bottom-up (synthesis) constraints of the domains. These modules are: Concept Formulation, for eliciting knowledge from a domain expert and structuring it into a concept map; Knowledge Analysis, to verify the concept map and cross check any inconsistencies found; Rule Generation, to produce a rule set for an expert system based on the concept map. A question dependency graph (QDG) supplies the control structures used in the concept formulation module. Control structures, defined a priori by knowledge engineers, are necessary to obtain the content knowledge from the experts. The utility of the QDG allows the control structures to be defined as needed and variations of the graph can easily address new applications.

Concepts and Base Facts

The basis for acquiring knowledge in KASH is organizing the cognitive processes an expert many formulate about a domain. The mechanisms to organize the processes must be tailored to the idiosyncratic representations created by the experts. However, experts tend to express their knowledge in unstructured and nondeterministic formats and difficulties arise when the knowledge must be converted into a reliable and useable format for expert system development. To properly address these representation problems, experts must be allowed, with minimal constraints, to describe their cognitive processes with respect to the intended application. Typically, the experts...
will define concepts that represent personal observations in the
domain to be modeled. Concepts are defined as symbols (text or
image) that capture the meaning or intention of objects and
events of an environment (Ausbubel, Novak and Hanesian,
1978). For example, an electrical system is an object, and boil
water in an event. In essence, a concept is a simplified and
generalized description of reality for a particular level of
abstraction (Novak and Gozin, 1984).

Concept formulation may be defined as the process of
extracting the characteristic features, termed criterial attributes,
of the objects or events. The criterial attributes are what uniquely
distinguish each concept (e.g., size, color, shape). A set of
common features and the degree to which the features are
accepted by the experts will determine the regularity (i.e.,
meaning) of a concept. The regularity implies precision on the
definition of the concept so that misunderstandings can be
minimized, see (Novak and Gozin, 1984). Criterial attributes
also allow concepts to be grouped or linked together to form
concept maps that can represent the conceptual and structural
knowledge of the domain experts. A concept map is simply a
hierarchical taxonomy of concepts. The hierarchy supports the
natural subsumption of concepts where broad concept
definitions are depicted at the top of the map and more detailed
and concrete concepts at the bottom. Concept maps promote the
reuse of concept definitions by allowing the use of existing
categories (e.g., dog) in different contexts (e.g., house, car).

Such a virtual feature is natural for multistep integration
because it provides support for a modular structure and the
development of a library facility of reusable concept definitions
across domains where applicable (McGraw and Westphal,
1988). The graphical nature of concept maps also allow
multistep conflict to be made explicit thereby excising the
knowledge structure of faulty linkages and misconceptions
(Westphal and Reeter, 1990).

The concept map in KASH has been extended to include
base facts. Base facts have their own definition because they are
fully instantiated (e.g., parts, ingredients, symptoms) and do not
require further justification for their existence. Base facts are
measurable and observable in the context of the concept and
support the criterial attributes of a domain. The criterial attributes
are necessary for distinguishing between concepts and
structuring questions into categories during concept formulation.
The criterial attributes also support the rule generation module
because they are variable entities (e.g., size is big, color is red)
that can be compared, contrasted, or combined with other
criterial attributes to form the rules produced by KASH.

### Question Dependency Graph

To facilitate the development of a concept map, the experts
must be interviewed using questions pertinent to the content and
structure of the problem domain. As Novak (1989) stipulates,
the sequencing of questions presented to the expert should be
tailored or grouped into specific sets of knowledge and the range
of these sets should address questions at broad superordinate
levels and become more narrow and precisely defined at the base
level. In KASH, the questions permissible to ask of the domain
experts during the development of the concept map are specified
by the knowledge engineers and represented in a question
dependency graph (QDG). The QDG has been abstractly based
on the six types of questions as defined by LaFrance (1987)
where each question type decomposes the QDG into a category
of queries (broad and specific) to be asked with respect to
categorical definitions. The six types are:

- **Grand Tour**, to identify domain and subdomain boundaries,
e.g., "What is the purpose of this system?"
- **Cataloging the Categories**, to support, organize, and
  structure the concepts, e.g., "Can the concepts be ordered?"
- **Ascertaining the Attributes**, to specify values and ranges of
  the criterial attributes for concepts and base facts, e.g.,
  "What values can attribute assume?"
- **Determining the Interconnects**, to define the causal
  relationships between concepts, e.g., "Does base fact
  support the concept?"
- **Seeking Advice and Cross Checking**, to compare and
  contrast information in the concept map, e.g., "Does concept
  contain any base facts?" or "What value of attribute
  is out of range?"

During the development of a QDG, a knowledge engineer
must specify the questions that will be presented to the domain
expert. The questions are defined to separate the control
knowledge from content knowledge of the domain (analysis or
synthesis). It is important for the knowledge engineer to
understand the basic structure of the domain to effectively define
questions, categorize the questions by type, and determine the
control sequence of the questions. Insufficient specifications at
this phase can lead to poor interview sessions with the domain
expert, who will supply the content knowledge (response to the
questions). In the QDG, the questions are represented as node
structures and are linked to other node structures to determine the
sequencing.

The node structure consists of a question frame, question
type, selection guide, and a variety of support attributes. The
question frame contains the actual text generated during a query.
Questions are developed by the knowledge engineers about the
control structure of the domain. The questions derived are used
to respectively refine or compose the concepts in an analysis or
synthesis domain. The six different question types defined
above are used to inscribe the scope of the questions are limited to
the current query category. By doing so, the focus of the experts
are concentrated on the particular task level being modeled. The
selection guide is satisfied when the knowledge engineer
specifies the response expected from the question frame with
respect to one (or several) of the system objects specified in
KASH. These objects are a prioritized taxonomy of concepts,
base facts, edges, attributes, and standard representation
mechanisms (i.e., integers, reals, text, etc) that help to resolve
which question to ask.

The edges form the explicit relationships between the node
structures and determine which tuples of questions may be
presented to the domain experts. A tuple of two questions is
interpreted to imply: if question one is asked, then question two
may be asked. There can be many-many relationships between
the node structures, thus forming a large combination of

102
questions tuples. The edges are coupled with edge weights to
determine the ordering of question preference - a form of
prefiltering. These weights may be displayed as a range of
numbers or as an alternative representation (i.e., low, med,
high).

A question dependency graph based on a diagnosis process. The order of
node letters and the high, medium, or low priorities indicates direction,
→ is used to denote explicit control.

Figure 2: Question Dependency Graph

Figure 2 shows an example of a partial question dependency
diagram for a circuit fault diagnosis. Each circle represents a
unique question (defined by a knowledge engineer) that may be
asked during a knowledge acquisition session. The box
represents the node structure (only one is detailed) for a
particular query scheme. The question name is Decompose
Concept(s), the question type (QT) is Grand Tour, the question
frame (QF) is defined as "What are the possible diagnoses of the
concept?" (variable terms in the question will get instantiated
during execution), the selection guide (SG) is expecting the
response of the domain expert to be a concept, and a
miscellaneous collection of support attributes helps to specify
display formats (DF) and the control strategy (CS). The links
between the nodes are labeled with high, medium, and low
priorities to indicate the control paths the system should pursue.
In this example node D is connected to nodes E and F. The
heuristics defined in the concept formulation module will select
the better candidate question (E or F) to ask based on the state
of the concept map.

A QDG editor is used for the construction of the question
diagrams. The editor allows the knowledge engineers to construct
specialized question bases through QDG link specifications.
New questions may be introduced into the graph at any time.
The set of base questions is always available, however it is the
format in which they are connected that will give a system its
functionality. This combination of nodes and links allow KASH
to be used across a large range of applications and proves to be
very versatile and powerful throughout the knowledge modeling
process. As the QDG changes, so will the system, because the
QDG is the canonical guidance referenced throughout the three
modules of KASH: concept formulation, knowledge analysis,
and rule generation.

Concept Formulation

Concept formulation, necessary to acquire the knowledge
structure from a domain expert, is interleaved between a top-
down and bottom-up elicitation strategy guided by the QDG.
The top-down refinements are focused on acquiring the abstract
concepts that are used to define the components of a domain.
These components are in turn supported by bottom-up facts that
represent the logical entities or extension of the concepts. The
use of top-down and bottom-up strategies in concept formulation
reflect the structures that exists in the different levels of
knowledge and the control nature of the analysis and synthesis
classes of problems. The analysis problems involve identifying
sets on objects based on their features. Synthesis problems
construct a solution from component pieces or subproblem
solutions. Figure 3 is a section of the top-down (and bottom-up)
process trace of a concept formulation based on the QDG for a
circuit fault diagnosis. The index (A-N) for each instantiated
question text shown (i.e., variables are bound within the context
of the question) corresponds to a node in the QDG shown in
Figure 2.

A) Define the terminology:
Top-down term(s) ⇒ diagnosis.
Bottom-up term(s) ⇒ symptoms.

B) What is the purpose of this diagnosis?
⇒ circuit fault.

C) Define any symptoms of a circuit fault.
⇒ manifold threads.

D) What are the possible diagnoses of a circuit fault?
⇒ control circuit failure.
⇒ sensor fault.

E) Can these be ordered?
⇒ no.

F) Is manifold threads a symptom of control circuit failure, sensor fault?
⇒ sensor fault.

G) Are there any other symptoms of sensor fault?
⇒ sensor mounts.

H) What are some attributes of sensor mounts?
⇒ connectors.

I) What value(s) can connectors of sensor mounts assume?
⇒ loose, broken, grounded, working.

J) What connections used in sensor grounding?
⇒ yes.

K) What value(s) of connectors would make sensor grounding succeed?
⇒ grounded, working.

L) What value(s) of connectors would make sensor grounding fail?
⇒ loose, broken.

M) Is this a default or inferred value?
⇒ inferred.

N) Can connectors be related to other attributes?
⇒ yes.

Figure 3: Question Trace from QDG

The concept formulation module will iteratively execute six
stages during the construction of the concept map. These stages
are responsible for selecting a concept from the graph,
establishing base facts for the concept, formulating a question to
apply to the concept, generating the question, then presenting
and accepting the results from the expert.

1 Assael (1978) addresses these abstractions in his propositional learning
theory.
Heuristic Criteria for Concept Selection - A concept is selected from the set of open concepts. This set represents those concepts that have not been fully explored by the QDG such that there are additional questions that may be asked about them. The concept is selected based on several factors including the order of importance that is explicitly specified by the user, a system defined depth first or breadth first selection mode, the recency of a concept definition, the level of attributes defined, the number of established links to the base facts, the number of derived (children) concepts, or a concept of enumerated type.

Establish Base Facts - The first step taken after the concept has been selected is to elicit the base facts that support the concept. These base facts define the fundamental units of the domain. The new base facts are specified by the user and supportive links are established to the selected concept. User selected base facts are then passed down to the selected concept from a parent concept. Note, parent concepts must resolve all supportive links before the concept map is considered complete.

Select Questions - The list of applicable questions for the current concept is calculated by obtaining all the edges attached to the previous question in the QDG and placing them into a list. A duplication filter is applied to the list to resolve any conflicts found in a list of previously asked questions maintained as a concatenation of the concept name and the question identifier. The list is then processed through some ordering filters that act on the edge weights, selection guide matching, and groups division types. The product of these filters results in a refined list of questions to be asked.

Generate Question - The product of the generate question stage is the selection of a single query node made from the refined list. This list of applicable questions for the current concept is acted on by a function of the question types and other system factors (not detailed here).

Question Presentation - After a single question has been selected it is presented to the user to solicit an answer. The control panel is consulted for the appropriate terms (top down/bottom up), display formats (bar graphs, text, pie chart, menu), and concept graph information (attributes, types, links).

Answer Questions - The expert responds to the question through either text (e.g., attributes, concepts, base facts, names, or values) or concept graph management facilities (e.g., graphics, node selection).

The output expected from the concept formulation module is a concept map representing a top-down or bottom-up process model. The concept map represents the observations of domain experts as guided by the control structure specified in the QDG. Figure 4 shows the concept map for the circuit fault example. If the execution of the module has not fully defined the range of the expert’s knowledge, the QDG may be reconfigured to accommodate the missing information. From this point the information gleaned from the expert in the form of concept modules and base facts is verified by both the system and the domain expert. Several structure analysis techniques, described below, have been defined to accommodate this task.

Knowledge Analysis - The knowledge analysis module is a hybrid of several interactive subsystems that are used to validate and refine the concept map. During validation, the knowledge acquired from the concept formulation module is analyzed to bring out inconsistent, incomplete, and unjustified information that may occur in particularly large systems or from the use of multiple experts. In refining the concept map, the system will look for ways to enhance the structure of the model through the application of a variety of techniques. Several proposed techniques are described below, also see Figure 5.

![Figure 4: Concept Map for Circuit Fault](image)

Cluster Analysis - Cluster analysis detects cases in which large numbers of concepts are derived from (or clustered around) a more general concept. It suggests that the general concept is too broad and it should imply several new concepts where the other concepts can be derived from, or some of the derived concepts should be promoted to the intermediate level concepts between the general concept and others. The introduction of intermediate concepts helps structure the knowledge elicitation stage and reduces the complexity of the acquired knowledge.

Entailment Analysis - The basis for entailment analysis stems from the attributes associated with the concepts. The technique makes the entailments between concepts obvious. There will be cases in which the entailments indicate that a concept can be subsumed by another concept because it is derived by another concept that has no other derivation.

Similarity Analysis - The knowledge engineer submits to the system a number of known cases for analysis. Each of these cases is represented in the format compatible with the acquired knowledge. The system will try to justify the given facts and results of the cases. If the facts and results cannot be explained, the system will seek help from the user by asking questions from the QDG focusing on the unexplained concepts and the failure.

Relative Analysis - Relative analysis is based on the reuse of concept modules in the graph. A concept associated with a base fact is selected for review. The analysis technique will propagate up the structure to a level outside the immediate hierarchy associated with the concept and query the expert, via the QDG, if it can be used in the adjacent paths. This type of analysis supports the use of virtual structures and serves as a form of memory cue entailment for additional structure refinements.

Subcomponent Analysis - A base fact is selected from an active path in the concept structure. This path consists of the base fact and all the concepts that comprise the path to the root node. The base fact is compared to each the concepts that subscribe to the path and questioned by the QDG for relevancy against the concept. Negative responses will indicate inconsistencies in the structure and will be resolved through further QDG examinations.
Conceptual Analysis - Attempts to enforce the principal that every base fact must be attached to a concept that is not further decomposed by additional concepts. If any such case is encountered it can be assumed that there are additional conceptual levels that have not been defined. Additionally, if a concept has been decomposed into another single concept, then the linear formation will invoke the analysis to collapse both into a unary concept structure.

Knowledge analysis can be interactive or selectable. In the first mode, interactive, a number of knowledge analysis techniques are selected from a menu before the concept formulation module is executed. As knowledge is acquired these techniques are automatically invoked to yield analysis results. The results are shown in a side window with the appropriate marks to indicate the different degrees of importance. In the second mode, selection, the knowledge analysis stage is entered after a knowledge elicitation session. The manner in which the analysis results are shown is the same as in the previous mode, but more thorough analysis techniques can be applied, particularly the similarity analysis. The topics of refinement and analyses to be performed are based on a prioritized scheme and will execute after any data changes or alterations to the concept structure are made.

First, the concept map is extended to include any terms (criterial attributes) to be used in the production of the rules. New attributes are created and manipulated by attribute->attribute, attribute->concept, and attribute->value questioning schemes defined in the QDG. The information provided by the experts in response to the questions specifies the attributes of a concept, the relationships of the attributes to other attributes, and the range of values the attributes may assume. Attribute definitions are initiated for each of the base facts in the concept map. The definitions are propagated to the concepts via an inverted inheritance scheme with question filters to terminate any attribute stream. The attribute relationships in the concepts represent rules. The rules are local to the concept for which they are defined and may be classified as either default, where new logical relationships between attributes are specified (i.e. set the temperature of material to equal the temperature of the gas), or inferred where the system must calculate the value of the used attribute (i.e., the temperature of the material must equal the temperature of the gas). The inferred values are present in the rule premises and the default values typically appear in the consequents.

Second, in the rule generation strategy the local rules are used by a series of high level top-down and bottom-up strategies that determine the direction in which final rules are developed. The top-down strategy employed in generating rules is based on a concept and the links to its supportive concepts. The bottom-up strategy directs the rule generation based on the lower level concepts to the higher level concepts that they support. This strategy supports both forward and backward chaining rule specifications for integration into the expert system.

Third, the characteristic or model of the target expert system shell language influences the different ways in which an optimal rule set may be generated. This depends on the different representation schemes, control strategies, deamons, triggers, etc. encountered in the different expert system shells. For example, instead of explicitly stating separate rules as would be necessary in a pure production system such as OPS-5, the number of rules generated can be reduced in an object oriented system such as NEXPERT or KEE due to the abstraction of frame structures.

Figure 6 shows the partial set of rules that have been created for the circuit fault diagnosis example. Before these rules are converted into a target expert system format they must be reviewed by a series of analysis techniques to ensure there are no redundant, conflicting, or circular rules. The final outcome is a knowledge base of objects, relationships, and rules that may be used in a variety of expert system applications.

Supportive Modules

There exists a number of external features that will be added to the basic functionality of KASH. These features are intended to extend support to the knowledge acquisition process and supplement the work of the QDG. The following represent a subset of the planned features to be developed:

- The time issues associated with a domain model will be addressed through a temporal subsystem. The concept graphs will be modified to accommodate the values necessary to represent the start, end, and latency periods. The question dependency graph will also be updated to elicit the necessary information. The temporal information will be represented through a time box consisting of a series of levels each partitioned into the respective intervals specified during the temporal questioning. This time box can be viewed from varying levels of detail depending on the concept graph depth.

- An enhanced control panel will contain a multitude of user defined parameters to coordinate the elicitation sessions. The panel will contain the terms used for the top-down and bottom-up definitions. Pre-logical settings for which top-down entries are best suited for the bottom-up selections will
be available. The search strategies (i.e., depth first, breadth first) can be set from the panel. A question list is defined in the panel and used to disable groups of questions for the session (usually low priority settings). Additionally, the analysis techniques can be made interactive or selectable from the panel settings.

- A history window will supply a meta-command interpretation of the events occurring in the system. This will provide documentational support to analyze the elicitation session as it progresses over time. The log will be used to archive the state of the concept model for easy reconstruction at a future date. Additionally the history list will be constructed for path resolution of the virtual system hierarchy.

- A glossary of terms defined by the expert during the elicitation of concepts and facts will be supported as a scrolling series of menus. This facility can support the comments associated with multiple expert development. The glossary support system can be expected to emulate a hyper-expert notation scheme where the terms defined can be a mixture of text and graphics.

```
IF level_of_voltage_test < 3
  THEN position_of_sensor_grounding = "open"
IF level_of_voltage_test >= 3
  AND level_of_voltage_test < 5
  THEN position_of_sensor_grounding = "closed"
IF connectors_of_sensor_mounts = "loose"
  AND cond_of_sensor_grounding = "poor"
  AND recommendation_of_sensor_grounding = "tighten"
IF connectors_of_sensor_mounts = "broken"
  THEN cond_of_sensor_grounding = "poor"
  AND recommendation_of_sensor_grounding = "replace"
IF connectors_of_manifold_threads = "dirty"
  THEN cond_of_sensor_grounding = "poor"
  AND recommendation_of_sensor_grounding = "replace"
IF connectors_of_manifold_threads = "corroded"
  THEN cond_of_sensor_grounding = "poor"
  AND recommendation_of_sensor_grounding = "replace"
IF cond_of_sensor_grounding = "poor"
  AND position_of_sensor_grounding = "open"
  THEN status_of_sensor_fault = "true"
  AND recommendation_of_sensor_grounding = recommendation_of_sensor_fault
  AND action_of_sensor_fault = "shutdown"
IF cond_of_sensor_grounding = "poor"
  AND position_of_sensor_grounding = "closed"
  THEN status_of_sensor_fault = "true"
  AND recommendation_of_sensor_grounding = recommendation_of_sensor_fault
  AND action_of_sensor_fault = "maintenance"
IF status_of_sensor_fault = "true"
  THEN status_of_circuit_fault = "true"
  AND recommendation_of_circuit_fault = recommendation_of_sensor_fault
  AND action_of_circuit_fault = action_of_sensor_fault
```

Figure 6: KASH Circuit Fault Rule Set

**Conclusion**

KASH is a general purpose knowledge acquisition shell that acquires information about a domain from an expert. Because KASH has been designed to support both analysis and synthesis applications, it may be applied to a broad range of systems including planning, design, classification, scheduling, decision support, and diagnosis where complex knowledge is often acquired from multiple domains experts. The three modules supplied by KASH provide the capability to acquire knowledge in context (i.e., customized for each domain): Concept Formulation, for structuring and eliciting the knowledge from a domain expert into a concept map; Knowledge Analysis, to validate the concept map; and Rule Generation, to produce a rule set for an expert system based on the concept map. All the information elicited from a domain expert is guided by a question dependency graph (QDG). The QDG is developed by a knowledge engineer to separate the control knowledge from the application knowledge. Thus, the QDG is reconfigurable for customization across applications and domain experts. Furthermore, KASH has the capabilities for producing a knowledge base that can be used to generate alternative rule sets for different expert system shells.

**Acknowledgements**

The authors would like to give special thanks to Joseph Novak for discussing his experience with concept maps; Ken Ford and Jack Adams-Webber for their support and ideas; Dick Weissblat, Karen McGraw, and Mike Bloom for their initial reviews. The views and conclusions contained in the paper are those of the authors and should not be interpreted as representing the official policies, either expressed or implied of their respective companies or governmental institutions.

**References**


A Graphically Oriented Automated Knowledge Acquisition Tool

Jennifer M. Sargeant, MSE and Joseph M. Schuerger, Ph.D.
Harris Corporation, GESD
Box 96000 - MS 20/9563
Melbourne, Florida 32902

ABSTRACT

Knowledge-based systems can be developed more effectively and accurately by automating the knowledge acquisition process. Knowledge Acquisition has historically been the major bottleneck in the development of knowledge-based systems. This paper describes an approach to automating the process via use of an Automated Knowledge Acquisition Tool (AKAT). This tool was developed under the Automated Knowledge Acquisition/Intelligent Authoring Tools (AKA/IAT) project. This was a tri-service effort with the Air Force Human Resource Laboratory, Army Research Institute, and the Naval Training Systems Center.

AKAT provides: (1) a format; (2) a structured process for subject-matter experts (SMEs) to build knowledge bases that are needed for the design of computer-based training systems; and (3) a set of on-line tools to assist, train and support the SME in the knowledge acquisition process. This PC-based artificial intelligence project focuses on reducing the largest single block of time required for the knowledge acquisition process - the knowledge engineering process.

INTRODUCTION

This project had two primary objectives: (1) to facilitate the extraction of expertise from a subject matter expert; and, (2) to provide a tool for diagnosing student performance errors.

The first objective was designed, developed and implemented by Harris Corporation. The second objective was subcontracted to Software Productivity Solutions (SPS), Inc. and identified as the Student Diagnostic Module (SDM). This paper will concentrate on the unique features of AKAT that facilitate the knowledge acquisition process, namely:

- Mouse-driven graphic interface for knowledge acquisition.
- User-friendly features that eliminate the need for a Knowledge Engineer.
- Adaptive help system.
- Use of metacognition concepts in tutorials.
- Use of Task Hierarchy to drive the KA process.

KNOWLEDGE REPRESENTATION

AKAT is designed to operate on the widely available 286 PC with EGA graphics, and requires the use of a three-button mouse. It was developed in Smalltalk/V286, an object-oriented language for the PC. AKAT uses a knowledge representation formalism that allows for the acquisition of task knowledge, both procedural and declarative.

Procedural Knowledge

Procedural knowledge is represented in the form of Modified Petri Nets (MPNs), which are constructed by the SME through a mouse driven graphical interface called the MPN Editor. An MPN consists of nodes which are connected together by arcs. A node can be a step or an event. A step represents an action, whereas an event represents the result of, or condition for a step (action). Figure 1 shows the MPN Editor which consists of an icon pane and a graph pane. The icons include step, event, arc, continuous step, pan, cut, paste, zoom, and scroll.

Declarative Knowledge

Declarative knowledge consists of supporting information or knowledge which can be attached to any step within the MPN. Five forms of Declarative Knowledge have been chosen to be included in this project. These include rules, facts, procedures, concepts, and other. Other is an arbitrary label for any form of miscellaneous declarative knowledge. Declarative knowledge is defined by the SME through the Node Editor. The Node Editor is a text-based editor which is accessed through the MPN Editor.

Task Analysis Data Collection

In addition to the procedural and declarative knowledge, AKAT provides a Task Analysis Data Collection capability. This capability is designed to collect data associated with individual MPNs, and provides additional information necessary to support the Task and Skills Analysis (TASA) report. The Task Analysis Data include:
• Task Analysis Descriptors - descriptors provide values for each of the following parameters: task criticality, frequency, level of difficulty, and duration for each MPN. These values range from -3 to +3 and provide verbal descriptors to help the SME define the parameter.

• Manpower - a simple spreadsheet interface is used by the SME to enter the type and number of personnel required to complete the task defined by the MPN. Manpower types include Officer (O1-O6), Civilian (GS1-GS12), and Enlisted (E1-E9).

• Documents - a variety of documents (e.g., Handbook of Maintenance/Operating Instructions) can be entered by the SME which will provide additional levels of detail on specific procedures.

• Equipment - a variety of equipment can be entered by the SME which describes tools, test equipment, etc. required to complete the task defined by an MPN.

Figure 2 shows an example of the task analysis descriptor for “Degree of Difficulty”.

KA TOOLS

In AKAT, SMEs are considered to be experts in their respective fields. They, however, are not experts in knowledge engineering and the knowledge elicitation process. AKAT aids the expert by providing a series of knowledge engineering toolsets which support the SME in the knowledge building process. These tools enable the SME to focus on building a domain by reducing the cognitive workload required for system interaction. These tools include an on-line help system, a tutorial system, and an automation of the Task Hierarchy building/revision process. A more detailed description of each of these tools follows.

Help System

AKAT’s help system is robust in that it provides a varied and adaptive means of providing help, that is based on the needs of the SME. Two forms of “help” are used: user-requested and system-initiated. User requested-help is represented in two different menus: Quick Help and Natural Language (NL) Menu. Quick Help is designed to be used by the SME when he has a specific question in mind. It presents a list of objects and
actions which answer the question of "Explain how to <action> <object>." The NL Menu, however, provides a structured form of natural language which allows the SME to ask many different questions, and also ask the same question in many different ways. The beginning phrases of an NL Menu include "What is a ...", "How can I ...", "Explain how to ..." and "Give an example of an ...". This form of help is particularly useful when the SME does not know exactly what question to ask. NL Menu allows the SME to build a question from a series of phrases.

Both Quick Help and NL Menu can be adapted to the needs of the particular user. The vocabulary and structure of the phrases can be modified, as can the text presented to the SME. Figure 3 shows a "NL Menu" query and corresponding help box produced by the query.

In addition, the help system provides "Help" on three different levels - beginner, intermediate, and advanced. The beginner level provides the most detailed explanation to a question, whereas the advanced help level would be less descriptive in answering a question. The Help System also provides system-initiated or "active help". This form of help is initiated by the system in response to actions performed by the SME. Active help is designed to be provided at the beginner and intermediate help levels.

Tutorial System

AKAT's tutorial system contains twelve tutorials which provide instruction on the three major functions of the KA process - naming, describing, and organizing. Figure 4 shows an example of the tutorial system interface. Metacognition concepts have been utilized to devise a systematic approach to: (1) enhance flexibility and individualization of AKAT tutorials; (2) allow instructional prescriptions that are more responsive to the needs of the individual SME; and (3) provides a means to increase the validity of the SME developed knowledge base. Additionally, AKAT provides a tutoring system that interacts optimally with SMEs that are knowledgeable in a way that is fundamentally different from the way it interacts with a confused SME. There are two types of tutorials - Effective Net Building (ENB) and System Interface. The Effective Net Building tutorials teach the SME how to build valid MPNs. Effective Net Building Tutorials include:

- AKAT Introduction - Introduces the SME to AKAT and MPNs.
- Walk-Through Demonstration - "Walks" the SME through the creation of a sample MPN (procedural knowledge).
- Walk-Through Summary - Provides the SME with a summary of the major topics covered in the Walk-Through Demonstration. SME can review some, all, or none of the review summaries.
- ENB Procedural Modules - Presents a set of modules which test the SME on the topics presented in the first three tutorials.
- Declarative Knowledge Introduction - Provides instruction on how to identify and describe facts, rules, procedures and concepts.
- Declarative ENB Modules - Presents a set of modules which test the SME's ability to differentiate between the different types of declarative knowledge.

The ENB procedural and declarative modules rate the SME on performance, and judgement-of-knowing estimates. The judgement-of-knowing estimate is a metacognition technique which asks an individual to rate how well they performed on the questions asked. Successful completion of a module is based on both performance and the SMEs judgement-of-knowing estimate for that module. The SME has three opportunities to complete a module. If the SME fails to complete a module the first time, prescriptive remediation is given and the SME is re-tested. If the SME fails to successfully complete a module a second time, both prescriptive remediation and feedback are given.

The System Interface Tutorials show the SME how to use the various tools and interfaces provided in AKAT. These tutorials provide instructional material on a "show-and-tell" basis. These tutorials include:

- Human-Computer Interface Tutorial - Explains basic concepts on how to use a mouse and basic windowing concepts.
- Main Menu Functions - Describes all of the functions provided by the Main Menu and its sub-menus.
- Help System - Shows the SME how to use the “Quick Help” and “NL Menus”.
- MPN Editor - Explains how to use the MPN Editor.
- Node Editor - Explains how to create and edit declarative knowledge attached to a step through the Node Editor.
- Task Hierarchy - Shows how to use the Task Hierarchy Interface.

**Task Hierarchy**

The Task Hierarchy is a graphic interface developed to assist the SME in the knowledge acquisition process. The Task Hierarchy provides a means for organizing knowledge and defining the inter-relationships between the MPNs in a graphical hierarchy. The Task Hierarchy consists of categories and MPN nodes. It allows SMEs to breakdown their expertise into categories, and to define the MPNs associated with each. The categories can be refined to any level of sub-categories. The Task Hierarchy allows the SME to define his/her expertise in either a top-down or bottom-up refinement methodology. The SME can also create and edit MPNs directly from the Task Hierarchy. A sample task hierarchy is depicted in Figure 5.

**CONCLUSIONS**

As Knowledge Acquisition moves from the research arena to more generalized applications such as AKAT, there is a need to develop generic KA tools that can be used by a wide variety of users. Computer-assisted knowledge engineering for KA is currently available in the commercial and research environments. These programs and their associated tools are normally used by knowledge engineers who have strong technical backgrounds. Typically, these programs use a knowledge engineer to facilitate both the extraction of knowledge from the SME and the building of the knowledge base. Therefore, the SME does not interact directly with the knowledge base. The AKAT project has been designed such that the knowledge engineering is embedded within the software and provides the users (SMEs with little or no computer literacy) with the necessary on-line tutorials and adaptive help features.
necessary to support the building of the knowledge base. The AKAT system also provides the users with a direct entry into the knowledge base development utilizing a set of tools that streamline the KA process, and targets the three primary components of the KA process: naming, describing and organizing.

The AKAT process utilizes embedded tutorials and performance aids and is based on advanced concepts from cognitive psychology, particularly in the area of metacognition (i.e., our knowledge about our own knowledge). These cognitive approaches have been used to enable learning and performance in laboratory settings. This project’s current development represents important pioneering applications of cognitive concepts to practical KA tasks. These cognitive approaches enhance flexibility and individualization of the AKAT tutorials, allow instructional prescriptions that are more responsive to the needs of individual user, and provides a means to increase the validity of the SME developed knowledge base.

Intelligent tutoring and aiding systems of the future will apply artificial intelligence and expert systems techniques to the development of highly individualized and powerful computer-based instructional systems. In order to provide effective training support for the military training mission, this technology must be able to significantly reduce the amount of time required by SMEs, instructors, and knowledge engineers in the creation, delivery and evaluation of training. The AKAT features discussed in this paper take a major step toward these future systems.

Acknowledgements

The authors would like to acknowledge the other AKAT design team members: Doug Lawson, Tim Lucas, Dr. Ron Morrison, and Rick Simonian.
KNOWLEDGE REPRESENTATION IN EXPERT SYSTEMS
USING REDUCED MATRICES TECHNIQUE

H. Lim and M. Schneider
Department of Computer Science
Florida Institute of Technology
150 West University Blvd.,
Melbourne, FL 32901.

ABSTRACT

This paper presents a new implementation technique in knowledge representation, the reduced bit matrix. It is an extension and a refinement of the conventional bit matrix technique. The objective is to develop a better technique that will represent a more precise internal knowledge and eliminate some of the shortcomings encountered in the bit matrix representation.

INTRODUCTION

Knowledge is a primary constituent of an expert system. Various techniques have been implemented to represent knowledge in that domain. A thorough description of these techniques has been compiled (Barr and Feigenbaum 1982). The three most common external representation schemes are the semantic networks, frames, and the production rules (Jackson 1986). Most of today's expert systems utilize production rules in representation of knowledge. This concept is introduced and popularized by Newell and Simon in 1972.

Production rules take the form of "IF premise THEN conclusion" condition-action pairs. The premise and the conclusion are often associated with some truth value called "certainty factor" (CF). The premise is a clause or a sequence of clauses with the

**information on exactly which clause in a conclusion of a rule is used to fired another rule in a knowledge base is known.**

Proceedings of the 3rd Florida Artificial Intelligence Research Symposium, 1990, by the Florida AI Research Society

© 1990 FLAIRS 0-9620-1732-9/90/0355-0112

112

conjunction operators AND, or OR. A negation operator NOT, can also be used preceding a clause. The conclusion may be a clause in a premise of other rules or a final conclusion (resolution).

In this paper, our focus is on the production rules knowledge representation scheme. There are several methods to implement this scheme in a computer program. Four internal representations are discussed in this paper: the tree structure, the bit matrix, the sparse matrix and the newly developed reduced bit matrix. Basic concepts, different inference procedures, and the pros and cons of each implementation are addressed. Examples are used throughout each scheme as comparison.

TREE STRUCTURE

A tree structure is a hierarchical data structure containing nodes that store knowledge, and arcs which connect the nodes. Clauses in the premises and conclusions from a knowledge base are represented as nodes in an AND/OR tree. If a clause plays a part in the occurrence of the other clause then an arc is drawn between them.

To further discuss this concept, an example is illustrated below:

Example - Let the knowledge base consists of the following 7 rules:

rule 1: if A and B then C
rule 2: if D or E then F and G
rule 3: if C and E then F and G
rule 4: if X then A
rule 5: if Y then B
rule 6: if F then S
rule 7: if G then R

Each alphabet in a rule represents a clause. The tree structure is shown below:

```
  C
 / \
A   B
/ \  /
A   B
/   /\   /
A   A   /\   /
/   /\   A   /
   /\   A   /
    /\   /\   /
     G   G   G
```

```
```
```
A rule is said to be a concluding rule if no arc emerges from the node (clause) of that rule. A rule that contains clause with no incoming arc is known as a starting rule. Thus, from the tree diagram, rules 6 and 7, which contain S and R respectively are the concluding rules. Rules that have clauses X, Y, D or Z are the starting rules and will require data from the user or any other input source.

Backward chaining and forward chaining are the two inference procedures applied in this data structure. In backward chaining, the system starts with a hypothesis and tries to establish facts which it needs to satisfy that hypothesis. However, forward chaining matches rules against facts to establish new facts until a goal is encountered.

Using the example above and the backward chaining inference procedure and assuming that the hypothesis is R and external data provided are X, Y, and Z, we have:

**Data In Blackboard**

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1) A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
</tr>
<tr>
<td>2) F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
</tr>
<tr>
<td>3) K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td>4) P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
</tr>
<tr>
<td>5) U</td>
<td>V</td>
<td>W</td>
<td>X</td>
<td>Y</td>
</tr>
<tr>
<td>6) Z</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td>7) E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
</tr>
</tbody>
</table>

The given data X, Y, and Z are stored in a data structure called the blackboard. In order to achieve the hypothesis R, clause G has to be satisfied. The inference procedure checks whether G is in the blackboard. If the blackboard contains G then R is concluded and the inference mechanism stops. If G is not present then a backward chaining process occurred whereby particular conditions (clauses) have to exist to fulfill G. This process propagates until certain conditions are satisfied from the data in the blackboard. The fulfilled conditions will then be added into the blackboard and a backtracking takes place. The * above indicates that A, B and E are satisfied through the external data and they are added into the blackboard. With the backtracking mechanism, clauses C then G are successively added to the blackboard and finally the hypothesis R is satisfied.

For the forward chaining, a starting rule is arbitrary chosen to begin the inference process. Suppose rule 4 is chosen to start the inference mechanism. As node X is present in the blackboard, A is added to it. This is continued with the firing of nodes B, C, E, F, G and consequently a goal S is obtained.

The tree structure is one of the natural way to represent knowledge. The advantages of applying tree structure in knowledge representation are:

1. The starting and concluding rules are easily identified. The starting rules consist of nodes with no incoming arc and the concluding rules have terminal nodes or the roots.

2. It is easily understood.

The drawbacks in this representation are:

1. We have to traverse the tree from level to level either by forward chaining or backward chaining to find a solution. Although a doubly linked list structure will save the traversing time, extra memory will be used.

2. Tree modification is difficult. When new rules are added to the knowledge base, they have to be inserted to the tree. The insertion involves complicated linking of correct nodes. Deletion of rule is even more difficult as not only nodes need to be removed from the tree but also all the arcs to that nodes.

3. With the inference procedures, the system always has to start from the concluding rule (the child node) in backwards chaining, or from the root node in forwards chaining.

4. Inference mechanism is very time consuming.

**Bit Matrix (BM)**

Bit matrix (Schneider and Kandel 1988) is a N by N square matrix with N as the number of rules in the knowledge base. The entry of this "rule-based Bit Matrix" is defined as follows:

\[
BM[i,j] = \begin{cases} 
-1 & \text{if the conclusion of rule } i \text{ can be involved in the evaluation of rule } j \\
0 & \text{otherwise}
\end{cases}
\]

113
This perfect matching constraint (valued 1) can be relaxed with the application of fuzzy reasoning (Schneider et al. 1987). The matching between a conclusion clause and a premise clause is made through fuzzy logic and a matched value M (between 0 and 1) is computed (Schneider and Kandel 1987), (Schneider and Kandel 1988), (Schneider and Kandel 1988). This value M is compared to a threshold value T which is preset according to the application of the expert system. The entry to the IBM can be rewritten as follows:

\[ r_{IBM[i,j]} = \begin{cases} \text{M} & \text{if } M \geq T \\ 0 & \text{otherwise} \end{cases} \]

Two significant information are observed following the construction of the bit matrix. One being that if the row contains all zeros then the i\textsuperscript{th} rule is a concluding rule. The other is when the column contains all zeros then the premise of i\textsuperscript{th} rule must come from the data.

The bit matrix is created when the expert system is invoked and is updated whenever a new rule is added into the knowledge base or a rule is deleted from the knowledge base. Applying the example earlier, the bit matrix is given as follows:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Rules 6 and 7 are concluding rules as rows 6 and 7 contains all zero values. Rules 4 and 5 need data from the blackboard since columns 4 and 5 also contains all zeros.

A different inference method known as direct chaining (Schneider and Kandel 1988) is applied. Direct chaining uses the bit matrix and a data structure to perform the inferences. The data structure consists of two linked lists: a list of fired rules (LFR) and a list of fired concluding rules (LFC). LFR contains rule numbers that participate in the inference procedure. LFC consists of rule numbers which besides of participating in the inference mechanism, are also concluding rules. The concluding rule is determined from the IBM.

Continuing with the previous example following the algorithm given in (Schneider and Kandel 1988) with data (X, Y, Z), the conclusion(s) can be obtained from the LFC list. LFR in this case is [2, 4, 5, 1, 3, 6, 7] and LFC is [6, 7].

There are numerous benefits in using bit matrix than any other techniques:

1. The explanation module is simple. To explain how a conclusion is derived, we first find the rule that fires that conclusion, say rule i. From the i\textsuperscript{th} column of the matrix, we can tell which rule constitute the firing of rule i. This is done recursively until all the rules and data which are involved in the firing of rule i are found.

2. It is easy to restart the system if it "gets stuck". Arbitrary rule is chosen to start the inference procedure and if the rule cannot be fired (stuck) a next rule is considered.

3. It tells which rule is intermediate or concluding as described above.

4. Inference mechanism is fast. It does not have to start from the root or beginning of a tree.

5. Implementation and tracing is simple.

6. Updating is simple. Insertion and deletion of a rule are easily done by just simply adding a row and a column to the matrix.

7. Entries in the bit matrix are not necessarily binary.

8. Validation process is easy. The matrix is self-multiplied. If at least one of the values in the major diagonal of the matrix is non zero then a loop appears in the knowledge base.

The disadvantages are:

1. It needs an enormous amount of memory.

2. It wastes a lot of memory space.

3. Only the rule itself and not its specific clause is known from the IBM.

SPARSE MATRIX

Most of the entries in a bit matrix are zeroes. This matrix is known as a sparse matrix (Harris and Kohout 1988). An enormous amount of memory is wasted to include the zero values in a matrix. The ideal would be not to physically store the zeroes and still be able to process the matrix as though the whole matrix is present. Sparse matrix representation normally takes the form of linked lists. Only
the rule numbers that have a non-zero entry in the bit matrix are on the lists.

Applying the example that is used throughout to this structure, we obtained the following figure:

```
1  3
  2  8
  1  4
  6  
  1
```

The advantages are:
(1). The most significant advantage is the huge amount of memory space saved. The worst case being each rule helps firing every single rule in the knowledge base.
(2). Using linked list instead of two dimensional array. Only those non zero entries are stored in the list.

The disadvantages are:
(1). Referencing is difficult. No indexing as can be found in the bit matrix.
(2). Much more time consuming compare to bit matrix representation.

REDUCED BIT MATRIX (RBM)

Instead of having rows and columns as the rule numbers as in rLB, a matrix associates a column to a clause in a premise and a row to a clause in a conclusion is built. This newly formed matrix need not be square and is denoted as "clause-based Bit Matrix" (cLB). The column is denoted as P_{n,i} while the row is represented by C_{i,j}. i indicates the rule number and j is the order of the clause occurred in the premise or conclusion of that rule.

The entry of the matrix is defined as follows:

\[ cLB(C_{i,j}, P_{n,i}) = \begin{cases} 1 \text{ if the } j^{th} \text{ clause} \\
                   \pi \text{ in a conclusion of rule } i \text{ can be} \\
                   \text{involved in the} \\
                   \text{evaluation of } n^{th} \\
                   \text{clause in a} \\
                   \text{premise of rule } m. \\
                   0 \text{ otherwise}
\end{cases} \]

This implementation specifies exactly which clause in a conclusion of a rule is used to fire which rule and also which clause in a premise needs to be provided. Similar modifications as described in the rLB can also be applied here to obtain a resultant value other than one (Schneider et al. 1987), (Schneider and Kandel 1987), (Schneider and Kandel 1988), (Schneider and Kandel 1988). However, additional memory space is required in this implementation. More zero entries appear in this matrix and substantial memory is wasted in storing all the zero values.

With this implementation, we can immediately recognize from the matrix which clause in a conclusion of a rule is used to fire which rule, and also, which clause in a premise of a rule needs to be provided. Applying this new matrix to the previous example, we obtained a matrix below:

\[
\begin{array}{cccccccc}
p_{1,1} & p_{1,2} & p_{2,1} & p_{2,2} & p_{3,1} & p_{3,2} & p_{4,1} & p_{4,2} & p_{4,1} & p_{4,2} \\
c_{1,1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
c_{1,2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
c_{1,3} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
c_{1,4} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
c_{1,5} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

This matrix is in turn being "reduced" to consist of only rows and columns which contain at least one non-zero entry. Large memory space is conserved as a consequence. Two array data structures are also created simultaneously as shown:

\[
\begin{array}{cccccccc}
q_{1} & 1.1 & 1.2 & 1.1 & 1.2 & 1.1 & 1.2 & 1.1 \\
cv & c_{1,1} & c_{1,2} & c_{2,1} & c_{2,2} & c_{3,1} & c_{3,2} & c_{4,1} & c_{4,2} \\
     & 1.1 & 1.1 & 1.1 & 1.1 & 1.1 & 1.1 & 1.1 & 1.1 \\
\end{array}
\]

This matrix is defined as the "clause-based Reducet Bit Matrix" or cLB. The "rule-based Reduced Bit Matrix" or rLB can also be produced by the same token. The two arrays are known as row values RV, and column values CV arrays. RV stores the \( p_{1,1} \) values of the \( P_{n,i} \)'s while CV keeps track of the \( i,j \) values of the \( C_{i,j} \)'s in the cLB.

Explanations that are used for the bit matrix are still applicable with minor modifications. A rule \( i \) is a concluding rule if none of the \( i,* \) values appear in the CV array.
Similarly, a clause in a premise \( p_{i,j} \) has to come from the data if the \( i,j \) value is not in the RV array. From the example, rules 5 and 7 are concluding rules while clauses D, Z, X and Y have to be provided.

The indication of which rule participates in firing another rule or which premise needs data is not as simple and noticeable as the one indicated in the rRBM. Indices in the cRBM no longer symbolize the rule numbers. In fact there might be no row/column or more than one row/column that represent a rule. Using similar example, we see that rule 4 appears in the fifth row and not in any column of the cRBM while rule 3 appears in rows 3 and 4, and also columns 3 and 4.

A search is performed to resolve this problem by going through the array and locating the position \( p \), where the given \( i,j \) value appears. Two cases occur in the search:

1. If no match occurs then \( p_{i,j} \) has to come from the data.
2. If there is a match then \( p \) will indicate the \( n^a \) column of the cRBM.

In the CV array,

1. If no match arises then rule \( i \) is a concluding rule.
2. If there is a match then \( p \) will indicate the \( n^a \) row of the cRBM. Let \( l \) be the column location in this row where there is a one entry, then with reference to \( n^a \) position of the RV array, a \( m,n \) value is obtained. This indicates that rule \( i \) participates in the firing of rule \( m \).

This search can be performed by a binary search with a time complexity in the order of \( O(\log n) \) (Starkey and Ross 1984).

The cRBM inherits all the benefits of the original bit matrix and more. Memory space is reduced in the cRBM as compared to the matrix described earlier in this section. There are not many disadvantages that occur in this new implementation. The disadvantages at the present moment is:

1. There is no longer a direct referencing to a rule in the RRM.
2. In order to find out which rule participates in firing another rule is more difficult but not unmanageable as indicated before.

CONCLUSION

Overall, the reduced bit matrix shows a much promising utilization for implementation for knowledge representation in terms of effectiveness, efficiency and completeness. The inference mechanism speed is greatly increased as the search time is reduced with this implementation.

REFERENCES

WEAK PARACONSISTENT LOGICS
FOR
KNOWLEDGE AND BELIEF

Roderic A. Girle
Automated Reasoning Project
Australian National University
Canberra ACT 2601
AUSTRALIA

ABSTRACT
Epistemic and doxastic logics are used as models for the machine as knower and believer. We look at a range of normal and non-normal logics, and ask about the plausibility of the pictures they present of an ideal epistemic/doxastic agent. In terms of four problems (logical omniscience, deductive omniscience, positive introspection, and negative introspection), the normal modal logics do not provide plausible models. Weak modal logics give more plausible models for the epistemic agent.
Semi-classical paraconsistent logics are better for the doxastic agent.

INTRODUCTION
There is a need in artificial intelligence to find a plausible model for the machine as an epistemic agent. One alternative has been to make use of epistemic and doxastic logics, which can be seen as providing models of the capacities of ideal epistemic and doxastic agents. Many researchers have accepted logics which are at least as strong as the S4 style logic advocated by Hintikka ([1962], cf. Lenzen [1978]), or as strong as an S5 based logic. But, these modal logics provide wildly unrealistic models of the capacities of epistemic agents (cf. Girle [1989b], Girle and McRobbie [1988], Scott and Jennings [1986], Girle [1973], Hocutt [1972]).

There are four obvious problems with such systems: logical omniscience, deductive omniscience, positive introspection (the KK-thesis), and negative introspection. Many recent papers begin by acknowledging the problems of logical and deductive omniscience, and many devices are used to ameliorate the problems while retaining an S5 logic. (Halpern [1986], Ginsberg [1987], Vardi [1988]) Little consideration has been given to the alternatives to the strong normal modal logics.

In this paper we do three things. We first look at a range of modal logics which could be interpreted for epistemic and doxastic purposes.

The logics will include the normal logics T, S4, S5 and the closely related D systems, and also S0.5, D0.5, the E systems, and the C systems. The logics will be compared in terms of the four problems set out above. It becomes clear that the normal modal logics are problematic for the modelling of epistemic agents, and that those problems can be reduced or disposed of by turning to weaker non-normal logics. The logical consistency of belief systems will be discussed.

Second, we consider, and set out a semi-classical logic (Dunn [1976], Girle [1982]). This logic is paraconsistent because it lacks the classical logic ex falso quodlibet thesis:

\[(p \land \neg p) \rightarrow q\]

Logics which lack ex falso quodlibet are more plausible than classical logics for managing contradictory belief.

Finally, we will raise some questions about what might be done to weak modal logics to make them even more reasonable as models for the epistemic/doxastic agent.

WAYS OF SEEING AN IDEAL AGENT
An epistemic logic can be seen as giving a model of an ideal agent. We can get a snapshot picture by looking directly at the axiomatisation of the logic (Girle [1989b]). We begin by looking at three normal modal logics: T, S4, and S5, and then at non-normal S0.5. These systems are related as follows:

\[S0.5 \subseteq T \subseteq S4 \subseteq S5\]

We start with epistemic S5 and work down through the systems, seeing what is lost at each point, maybe to advantage. We then look at the parallel sequence of E systems. For doxastic logics we will start with D4 and work down through the D systems, and then look at the parallel C sequence.

The S5 Agent
Consider a Lemmon [1966] style axiomatisation for an epistemic S5. The letters P and Q are formula schema. \( \tau \) is for \(( p \rightarrow p \). Epistemic S5 has the axiom schema of any classical propositional logic together with:

1. \( Kp \rightarrow p \)
2. \( K(p \rightarrow q) \rightarrow (Kp \rightarrow Kq) \)
3. \( Kp \rightarrow KpKp \)
4. \( \neg p \rightarrow Kp \neg Kp \)
5. \( Kp \rightarrow KpKp \)

Proceedings of the 3rd Florida Artificial Intelligence Research Symposium, 1990, by the Florida AI Research Society
c. 1990 PLAIRS 0-9620-1752-9/90/0350-0117

117
The Rules of inference are:
\[ R1. \quad \neg P, \neg (P \rightarrow Q) \vdash \neg Q \]
\[ R2. \quad t \Rightarrow \neg K_P t \]
\[ R3. \quad \neg P \Rightarrow \neg K_P P \]

\( A5. \) and \( R2. \) are redundant for \( S5. \) but it is useful to note them for later comparisons. The axioms and rules of inferences are often referred to by the following names:

\[ A1. \quad \text{Veridicaly} \]
\[ A2. \quad \text{Distribution} \]
\[ A3. \quad \text{S4-axiom or KK-thesis or Positive Introspection Thesis} \]
\[ A4. \quad \text{Strong S5-axiom (Platonic Axiom)} \]
\[ A5. \quad \text{Weak S5-axiom or Negative Introspection Thesis} \]
\[ R1. \quad \text{Modus Ponens} \]
\[ R2. \quad \text{Weak-necessitation} \]
\[ R3. \quad \text{Necessitation or Full Necessitation} \]

We note that the Veridicaly Axiom is uncontroversial. So we move on to consider other matters.

First, consider Necessitation. The \( S5. \) ideal agent knows all the theses of both first order logic and epistemic logic. This is strong logical omniscience.

Consider then the Distribution Axiom. This axiom, when put together with Necessitation, gives an \( S5. \) agent who is logically and deductively omniscient in the strongest sense. The agent draws all the logical conclusions that follow from what is known.

We now turn to the \( KK-thesis. \) This axiom means that, if we have a univocal account of an active and aware sense of knowing, then it follows that the agent modelled here is a fully self-aware knower. Such an agent cannot truthfully assert, for example, that they know all along that their partner was being unfaithful, but did not know that they knew it. There is no room for self-deception. On the other hand, if the account is of some minimal sense of knowing, the account will lack a vital element, and will need to be part of a multiply modal epistemic logic. If the multiply modal system is composed of two \( S5. \) logics, then we will still have the fully aware knower in an \( S5. \) model.

We turn next to the strong \( S5. \) axiom. The full force of this axiom is often avoided by considering only the weak \( S5. \) axiom. The weaker axiom seems, in a sense, more reasonable. But we cannot have the weak without the strong in \( S5. \)

The ideal \( S5. \) agent is therefore, a fully aware knower who has immediate access to all the consequences of their knowledge, to all the theorems which constitute the logical structure of knowledge, and knows what they do not and cannot know. Knowledge will, for the \( S5. \) knower, constitute a maximal consistent theory, totally accessible. It is arguable that the only possible \( S5. \) knower is an omniscient God.

\textbf{The S4 Agent}

The ideal \( S4. \) agent is the knower who conforms to Hintikka's epistemic logic. The \( S4. \) agent is almost the same as the \( S5. \) agent, except for the absence of the \( S5. \) axioms.

It is possible for the \( S4. \) agent not to know that they do not know something. For example, if \( X \) were an \( S4. \) agent then \( X \) could believe that they knew that \( P \), but \( P \) could be false, and so \( X \)'s belief would be a false belief. So, the \( S4. \) agent is permitted to have false beliefs about what they know. This is in contrast to the \( S5. \) agent who cannot have false beliefs about what they know.

\textbf{The T Agent}

The \( T \) agent is fully aware of the structure of knowledge without necessarily being fully aware of everything they know. The \( T \) agent knows all the logical consequences of what they know, but knowledge of contingencies may be unknown knowledge, implicit knowledge.

\textbf{The S0.5 Agent (Leemon's ideal)}

The \( S0.5. \) model is Leemon's ideal agent [1959]. The model contains only the weak Necessitation Rule. The \( S0.5. \) agent, like the \( T \) agent, is not necessarily a fully aware knower. Indeed, there are no theorems in epistemic \( S0.5. \) of the form \( K_P K_P \alpha \). So, if being self-aware is to be represented by formulas of the form \( K_P K_P \alpha \), then self-awareness for the \( S0.5. \) agent is a purely contingent matter. The \( S0.5. \) agent knows all the theorems of first order logic, but may not know that they know them. The \( S0.5. \) agent is an ideal \textit{Cartesian} agent (Descartes pg 191).

In summary, having looked at the four epistemic logics based on the \( S \) systems, we can see that these models for ideal epistemic agents have two things in common. The agents are deductively omniscient, and logically omniscient: with respect to at least first order logic. In view of this, these logics do not provide models for machine knowers related, except quite indirectly, to our notion of knowledge.

\textbf{The E Agents}

There is a set of \( E \) systems running in parallel, as it were, to the \( S \) systems we considered above. These systems lack any thesis of the form \( K_P \alpha \). The systems are all non-Cartesian. In these \( E \) systems, the principal gain is in the loss of the logical omniscience capacity.

The \( E0.5 \) agent is by far the most realistic of the \( E \) agents. But there are even weaker systems, and, as Richard Sylvan writes, "Philosophical virtue lies in weakness." (pg 3 [1986])

\textbf{Doxastic Parallels}

The \( S \) systems have closely related doxastic systems, or \( D \) systems. The doxastic syntax is gained by replacing the knowledge operator by a belief operator, as in: \( B_P \). The systems are gained in each case by replacing Veridicaly by the axiom known as the Consistent Belief Principle:

\[ A8. \quad B_P \neg P \vdash \neg B_P \neg P \]

Such doxastic logics suffer from all the relative faults of their \( S \) system cousins, particularly from logical and deductive omniscience.

Furthermore, they all have the logical consistency property. The agents modelled by these systems have logically consistent belief sets. This ideal is not obviously desirable. It certainly conforms to one view of what it is for an agent to be a rational agent. But it is not clear that logical consistency is a necessary condition for rationality.
If we modify the S systems by simply dropping Veridicality then the logics we get are Lemmon's C systems. Of even greater interest are the results of modifying the E systems by dropping the same axiom.

It is important to consider doxastic systems in which ex falso does not hold. There are semi-classical systems around RM in which ex falso does not hold, and for which there is a fairly straightforward semantics (Dunn [1976]). So we now turn to the semantics for the weak modal systems E0.5 and C0.5, based on both classical and semi-classical systems.

Semantics for Weak Modal Logics

It is useful in the long run, especially for logic programming, to set out the semantics for these systems in Hintikka's (1962) model-set-model-system style semantics (cf. Girle [1989a]). We begin with the semantics for classical S0.5 and E0.5 under epistemic interpretation and for D0.5 and C0.5 under doxastic interpretation. The set of formulas defined in the usual way, is denoted by WF (Well-formed Formulas).

For any logic, L, an L-model-system, Ó, is a set of models sets of WF, the model-sets being the field of a dyadic relation, M. If i M j, i is said to have (modal) access to j. The membership of the model-sets in an L-model-system is stipulated by a set of consistency conditions drawn from the conditions below (where µ, , µ, ... are for any model-sets in the model-system, and the elements D, E, F, ... are formula schema.) Each model-system, Ó, may be partitioned into two mutually exclusive and exhaustive sub-sets of model-sets, A and N, such that:

A = {µ | (X i ∈ Ó and X i M µ)}

A is the set of accessed sets, and N is the set of non-accessed sets. The set of classical propositional logic conditions, CPL, can be retrieved from Hintikka [1962]. The epistemic (modal) conditions are:

(C. -K) If (¬K X D) ⊆ µ, then (P X ¬D) ⊆ µ.
(C. -P) If (¬P D) ⊆ µ, then (K X ¬D) ⊆ µ.
(C. K) If (K X D) ⊆ µ, then (D) ⊆ µ.
(C. K+) If (K X D) ⊆ µ i, then (D) ⊆ µ i.
(C. F) If (F X D) ⊆ µ i, then there is at least one model-set, say j, in the model-system, such that (D) ⊆ µ j and M µ j, and µ i ⊆ A.
(C. K F) If (K X D) ⊆ µ i, then (D) ⊆ µ i.
(C. F+) If (P X D) ⊆ µ i, then there is at least one model-set, say j, in the model-system, such that (¬D) ⊆ µ j and M µ j, and µ i ⊆ A.

Let XS0.5 = (C. -K), (C. -P), (C. K), (C. K+), (C. F+), (C. F).

The conditions for S0.5 and E0.5 are:

C.S0.5 = CPL U XS0.5
C.E0.5 = C.S0.5 U {(C. K+), (C. F+), (C. F)}

The definitions of Validity are:

PL-valid(D) iff there is no µ i in any PL-model-system such that (¬D) ⊆ µ i.

E0.5-valid(D) iff there is no µ i in any E0.5-model-system such that (¬D) ⊆ µ i.

S0.5-valid(D) iff there is no µ i in any sub-set, N, of any S0.5-model-system such that (¬D) ⊆ µ i.

A validity test can be set out in a refutation tree which begins with the assertion that the negation of a formula is in an appropriate model-set or model-system.

For the belief logics there is a set of conditions, from which the sets for the systems are drawn, in which the K X is replaced by B X and in which the P X is replaced by C X. The conditions are appropriately named, and are augmented by:

(C. B C) If (B X D) ⊆ µ i, then (C X D) ⊆ µ i.
(C. B C+) If (B X D) ⊆ µ i, then (C X) ⊆ µ i.

The conditions for D0.5 and C0.5 are:

C.C0.5 = C.PL U {(C. -B), (C. -C), (C. B C), (C. B C+), (C. C)}
C.D0.5 = C.C0.5 U {(C. B C)}

The definitions of Validity are:

D0.5-valid(D) iff there is no µ i in any sub-set, N, of any D0.5-model-system such that (¬D) ⊆ µ i.

C0.5-valid(D) iff there is no µ i in any sub-set, N, of any C0.5-model-system such that (¬D) ⊆ µ i.

A Paraconsistent Epistemic Logic

We extend the syntax and semantics as follows. The set of formulas is denoted by WFR (Well-formed Formulas of RM and nearby logics). These formulas are extended by the addition of four flags: t, nt, f, nf. The intuitions semantics of these is of at least true, not true, at least false and not false respectively. We define the set WFR (flagged WFR) in the usual way. In the semantics for F*, a formula may be just true, just false or null value.

A KF*-model-system is a set of model-sets of WFR, the model-sets being the field of two dyadic relations, R and M. If i R j, i is said to have relevant access to j. If i M j, then i is said to have (modal) epistemic access to j. We begin with the conditions for the paraconsistent system, F*:

(C. t) (t D, nt D) is not a sub-set of any model-set.
(C. nt) (t D, nt D) is not a sub-set of any model-set.
(C. Not Both) (t D, f D) is not a sub-set of any model-set.
(C. t ¬) If (¬D) ⊆ µ i, then (f D) ⊆ µ i.
(C. nt ¬) If (¬D) ⊆ µ i, then (nf D) ⊆ µ i.
(C. f ¬) If (¬D) ⊆ µ i, then (t D) ⊆ µ i.
(C. nf ¬) If (¬D) ⊆ µ i, then (nt D) ⊆ µ i.
(C. tE) If (t D, E) ⊆ µ i, then either (nt D) ⊆ µ i or (nt E) ⊆ µ i.
(C. ntE) If (nt D, E) ⊆ µ i, then either (t D) ⊆ µ i or (t E) ⊆ µ i.
(C. fE) If (f D, E) ⊆ µ i, then either (nt f) ⊆ µ i or (nt E) ⊆ µ i.
(C. nfE) If (f D, E) ⊆ µ i, then either (nt D) ⊆ µ i or (nt E) ⊆ µ i.

The conditions for S0.5 and E0.5 are:

C.S0.5 = CPL U XS0.5
C.E0.5 = C.S0.5 U {(C. K+), (C. F+), (C. F)}
If we do so, then we have to abandon a classical material implication → and move to a non-classical →, and also move to different modal operators. Systems with the weaker Distribution principle have been formulated.

The semantics are as for KF** with the following qualifications. The epistemic conditions are the same as those above for any μj ∈ A. Changes are made to the conditions for members of the model-sets in A:

\[(C_1 \Delta^{+} \alpha) \text{ if } \{t \kappa \} \subseteq \mu_j \in A, \text{ and either } \{n \kappa \} \subseteq \mu_i \in A, \text{ or } \{s \kappa \} \subseteq \mu_i \subseteq \mu_j.\]

\[(C_2 \Delta^{+} \alpha) \text{ if } \{t \kappa \} \subseteq \mu_i \subseteq \mu_j, \text{ and either } \{n \kappa \} \subseteq \mu_i \in A, \text{ or } \{s \kappa \} \subseteq \mu_i \subseteq \mu_j.\]

\[(C.\text{Not Both}) \text{ is replaced by:}\]

\[(C.\text{Not Non}) \{n \Delta \} \subseteq \text{not a subset of any model-set.}\]

The epistemic conditions are:

\[(C.1 \Delta^{+} \alpha) \text{ if } \{t \kappa \} \subseteq \mu_i \subseteq \mu_j, \text{ in the model-system, such that } \{t \Delta \} \subseteq \mu_i \subseteq \mu_j, \text{ and either } \{n \kappa \} \subseteq \mu_i \subseteq \mu_j, \text{ or } \{s \kappa \} \subseteq \mu_i \subseteq \mu_j.\]

\[(C.2 \Delta^{+} \alpha) \text{ if } \{t \kappa \} \subseteq \mu_i \subseteq \mu_j, \text{ in the model-system, such that } \{t \Delta \} \subseteq \mu_i \subseteq \mu_j, \text{ and either } \{n \kappa \} \subseteq \mu_i \subseteq \mu_j, \text{ or } \{s \kappa \} \subseteq \mu_i \subseteq \mu_j.\]

\[(C.s \Delta^{+} \alpha) \text{ if } \{t \kappa \} \subseteq \mu_i \subseteq \mu_j, \text{ in the model-system, such that } \{s \Delta \} \subseteq \mu_i \subseteq \mu_j, \text{ and either } \{n \kappa \} \subseteq \mu_i \subseteq \mu_j, \text{ or } \{s \kappa \} \subseteq \mu_i \subseteq \mu_j.\]

\[(C.m \Delta^{+} \alpha) \text{ if } \{t \kappa \} \subseteq \mu_i \subseteq \mu_j, \text{ and either } \{n \kappa \} \subseteq \mu_i \subseteq \mu_j, \text{ or } \{s \kappa \} \subseteq \mu_i \subseteq \mu_j.\]

\[(C.\text{Not Both}) \text{ is replaced by:}\]

\[(C.\text{Not Non}) \{n \Delta \} \subseteq \text{not a subset of any model-set.}\]

The definition of Validity is:

KF** valid(D) iff there is no μj in any KF** model-system such that {n \kappa \} \subseteq \mu_i.

An analogous belief system can also be formulated.

IMPROVING THE MODELS

The Distribution axiom is one of the main "drivers" of deductive omniscience. But, if it were simply dropped, then the attempt to use logical models for knowledge and belief becomes questionable. But, consider replacing the Distribution axiom with the axiom:

A9. K\( \kappa (P \rightarrow Q) \rightarrow (K\kappa P \rightarrow \neg K\kappa \neg Q)\)
REFERENCES


KNOWLEDGE REPRESENTATION IN FUZZY RELATIONAL DATABASES

R. Joshi and M. Schneider
Computer Science Department
Florida Institute of Technology
150 W. University Blvd.
Melbourne, Florida 32901.

Phone # (407) 788-8000 Ext. 7280

ABSTRACT

Knowledge acquisition and knowledge representation techniques used by the databases are briefly discussed in this paper. We also discuss some of the features of an intelligent database capable of handling imprecise (inexact) or vague data. The database, denoted as the Fuzzy Relational Database-II, is designed for use with expert systems, is used to define various kinds of fuzzy attributes, fuzzy sets, similarities and links. This system incorporates fuzzy set theory combined with the theory of possibility for the representation of vague (fuzzy) information. The unique feature of this system is its capability to deal with fuzzy sets of the second order, and the ability to represent a fuzzy set in a graphical form.

INTRODUCTION

A question that comes to our mind when we hear the words knowledge, data and information is, “What is the difference between them?” Data is factual information used as a basis for reasoning, discussion and calculation. Data refers to numerous information suitable for computer processing. Knowledge is integrated information including facts and their relations, which provide a mental picture of some aspects of the world. It is the condition of knowing something with familiarity gained through experience or association. Level of abstractions in systems is similar to differentiation in calculus. These levels are used to study the changes within a group of heterogeneous objects so as to understand and make use of their patterns to control the new objects of each type. Knowledge, therefore, can be considered as data at a high level of abstraction. The amount of data necessary to characterize a piece of knowledge can be infinite in size, hence the meaning of a piece of knowledge can be imprecise or uncertain. In general, knowledge can be considered as a compact and sometimes imprecise way of representing a body of data. Information, on the other hand, is the communication or reception of knowledge, perhaps, using signals and symbols representing data.

Knowledge engineering refers to the methods, algorithms, and systems for the design, utilization and maintenance of data and knowledge. It can be considered as a study related to the computer-aided management of knowledge. Early processing systems were small in scale and their modelling, design, and management could be handled by one or more experts. The ever increasing complexity of applications and increasing need to capture and manage abstract knowledge require the collective effort of a large number of experts. Hence knowledge, and in turn data, can no longer be handled by individuals alone and its management is treated as an engineering discipline. Knowledge acquisition is the foremost step in knowledge engineering applications, the problems associated with it can be classified according to the following three attributes (Ramamoorthy 1989):

- Completeness of knowledge. The acquired knowledge can be complete or incomplete, depending on which suitable method is selected to solve the problem. Sometimes the amount of knowledge may be extremely large, and in this case a restricted set of knowledge is used to solve the problem.

- Accuracy of knowledge. If the knowledge acquired from the environment is exact then the data characterizing this piece of knowledge can be represented numerically or logically. In situations when the knowledge is not exact, it is almost impossible to enumerate or represent all the knowledge. Hence, before solving the problem, heuristics are applied to define a finite number of possibilities so that what is available can be treated as exact, after which a solution to the problem may be obtained in the given solution space.

- Knowledge about the objective. An objective which is well defined is represented in terms of measurable parameters of the problem environment, which makes it feasible to compare the quality of the various solutions. When the objective is poorly defined, the consideration is restricted to the measurable parameters and the objective is redefined as a function of these parameters, before attempting to solve the problem.

Various knowledge representation schemes feature different tools for knowledge and data acquisition and its management. These tools, however, need to be modified to suit a particular application and its computational environment.
there are many different methods of knowledge representation which include semantic networks, frames, etc., the major problem is the lack of an overall technique to guide the evaluation and selection of a representation scheme. Representation of knowledge at a higher level is influenced by the applications, while at a lower level it is influenced by the technological constraints. Therefore a representation scheme must be decided upon before developing the necessary hardware and software support mechanisms. To represent knowledge, we can use an existing conventional scheme such as one that is declarative, procedural or object-oriented, or we can design a new scheme to incorporate uncertain, incomplete, or fuzzy knowledge. The representation scheme may also have to support applications in a sequential processing environment or a parallel environment.

One issue of knowledge representation which is not adequately addressed is the representation of fuzzy knowledge, that is, the knowledge which is lexically imprecise or uncertain. The conventional approaches to knowledge representations lack the means for representing the meaning of fuzzy concepts. It does not provide an appropriate conceptual framework for dealing with representation of common sense knowledge, since such knowledge is by its nature fuzzy and noncategorical. In a general setting knowledge may be viewed as a notion of propositions. To constitute knowledge, a proposition must be understood which implies that meaning and knowledge are closely interrelated. Therefore we can say that in fuzzy logic, knowledge representation is based on test score semantics. The basic idea underlying test score semantics (Zadeh 1989b) is that a proposition in a natural language may be viewed as a collection of elastic or equivalently fuzzy constraints. For example, the proposition John is tall, represents an elastic constraint on the height of John.

**FRDB-II DESCRIPTION**

The Fuzzy Relational DataBase-II (FRDB-II) system model presented here utilizes second order fuzzy sets. It is developed to handle incomplete, imprecise, uncertain or vague information. FRDB-II is based on the classical relational database design combined with fuzzy set theory (Zemankova and Scharlak 1983), as the basis for the theory of possibility (Zemankova and Scharlak 1983), fuzzy logic, semantics and linguistics. Some of the outstanding features of the FRDB-II include possibility distribution, fuzzy relations such as Similitude and Link, possibility measures, translation rules, advanced query syntax, semantic power-tubility to express linguistic modifiers such as very, almost, etc., and above all its capability to deal with data of type range (intervals). Since fuzzy sets of second order facilitate the representation of data of type range, FRDB-II uses these sets to represent imprecise data. FRDB-II system satisfies the need for individualization; it takes into consideration individual differences in data perception. For example, a person from a community where the average height of the persons is 6 feet might say that any person with height less than 5 feet is definitely short. On the other hand if the average height of persons in a community is 5 feet, then anyone who is 5.5 feet in height will positively be treated as a tall person. In other words, a definition of fuzzy set TALL may differ from user to user although the actual height data in the database remains the same. The various aspects of vagueness in knowledge, which FRDB-II is capable of representing are:

- Inherent vagueness of all classification terms.
- Uncertainty of the meaning of vague terms due to statistical variability in communication.
- Variability in cognition, may be present due to sociocultural differences.
- Uncertainty of linguistic which prevails in sentences such as This is not very fast, I almost agree and It is almost fair. Such statements are very ambiguous; a statement as I almost agree does not indicate whether I completely agree or how much I agree. The perception of the meaning of the quantifier agree may vary from person to person.

Human beings are known to apply a threshold, analog variable in coming to a binary decision, and this threshold seems to be associated with personality factors and individual traits. This idea, with the help of fuzzy sets and the degree of membership, is used to handle imprecise information. The theory of possibility reflects that much of the reasoning is possibilistic in nature. The imprecision that is intrinsic in natural languages is handled using the concept of possibilistic distribution. Fuzzy set consists of objects and their respective grades of membership in the set, which is given by a membership function. Consider an example of a fuzzy set C labeled as PLEASANT (temperature) (Kandel 1984), where, C = {set of temperatures ∼ t}, with the membership function as,

\[ m_c(u) = \exp\left(-\frac{(u - t)^2}{b}\right) \text{ for all } u.\]

The parameter b provides the degree of fuzziness of the data to be represented. Let the membership functions given by a Floridian and a Czechoslovakian, respectively as follows,

\[ m_{c1}(u) = \exp\left(-\frac{(u - 74)^2}{6}\right) \text{ and,} \]

\[ m_{c2}(u) = \exp\left(-\frac{(u - 68)^2}{12}\right). \]

Fig.1 illustrates the above membership functions in a graphical form. It can be seen from the two membership functions that the Floridian not only likes warmer temperatures but also have a low tolerance for deviation from the preferred pleasant, temperatures of 74°F, whereas the person from Czechoslovakia considers 68°F, as ideal and accepts a broader range of temperatures as pleasant. This method of representing an imprecise knowledge about pleasant, satisfies the need of individualization.
FRDB-II, an extremely user-friendly, has four main modules: DEFINE, MODIFY, QUERY and SAVE. FRDB-II maintains the active database in the memory for the sake of speed and hence the user has the option to save/download the database onto a secondary storage media using the SAVE module. However, the system will automatically do the same at the end of the FRDB-II session.

The DEFINE, like the other modules, is highly interactive. It provides the user with the capability to define attributes - may be defined to be a record structure, *fuzzy sets* - may be defined over a particular range (intervals), - *similarities* and *link*. The fuzzy sets and attributes can be defined as a function of other fuzzy sets and/or attributes. The powerful capability of being able to represent inexact data as intervals which contributes towards the immense flexibility of FRDB-II in representing imprecise data, is a really unique feature.

Consider a non-fuzzy proposition, \( X \) is an integer, in the interval \([0,5]\). The interpretation of the proposition asserts that
1) It is possible for any integer in the interval \([0,5]\) to be a value of \( X \).
2) It is not possible for any integer outside the interval \([0,5]\) to be a value of \( X \).

In other words the proposition induces a possibility distribution, which means that if \( X \) is such that \( 0 \leq X \leq 5 \) then the possibility associated with the statement 1 above is 1. The possibility is 0 if \( X \) is not in the specified interval. Thus we see that in non-fuzzy propositions, possibility has a boolean value, either 1 or 0.

Now we fuzzify the proposition to \( X \) is a small integer, in the interval \([0,5]\). This piece of knowledge will be represented in FRDB-II by defining a fuzzy set \( \text{SMALL}_{\text{INT}} \) as follows:

\[
\text{SMALL}_{\text{INT}} = \frac{1.0}{0} + \frac{1.0}{1} + \frac{0.9}{2} + \frac{0.7}{3} + \frac{0.5}{4} + \frac{0.2}{5}
\]

where \( \frac{0.7}{3} \) signifies that the grade of membership of the integer 3 in the fuzzy set \( \text{SMALL}_{\text{INT}} \) is 0.7. Alternatively, this means that the compatibility of statement that \( 3 \text{ is a small integer, is 0.7, (Zadeh 1985a).} \)

Moreover FRDB-II also allows a fuzzy set to be defined over a range. For example we can define the fuzzy set \( \text{SMALL}_{\text{INT}} \) over the interval \([0,100]\) as,

\[
\text{SMALL}_{\text{INT}} = \frac{1.0}{0.10} + \frac{0.9}{11.20} + \frac{0.5...0.7}{21.50}
\]

which will indicate that the grade of membership of any integer between 11 and 29 is 0.9. The grade of membership of any integer between 21.50 is 0.5...0.7. The knowledge that *Jim's hair is reddish-brown*, could be represented as,

\[
\text{HAIR} = \frac{0.1}{\text{brown}} + \frac{0.5}{\text{red}}
\]

where HAIR is an attribute and the set of COLORS is (blond, brown, red, black).

Consider the following example of a fuzzy set \( \text{GROWTH} \) defined over a range which is divided into several intervals, each having a value of type range.

<table>
<thead>
<tr>
<th>RANGE</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.5</td>
<td>1.0..2.5</td>
</tr>
<tr>
<td>5..9</td>
<td>2.5..3.0</td>
</tr>
<tr>
<td>10..15</td>
<td>3.0..4.5</td>
</tr>
<tr>
<td>16..30</td>
<td>4.5..6.3</td>
</tr>
<tr>
<td>31..95</td>
<td>5.2..6.1</td>
</tr>
</tbody>
</table>

This kind of definition can be interpreted as, the fuzzy set \( \text{GROWTH} \) is defined over the range 1..95 years of age. The growth of a person at the age of 1.5 is 1.0..2.5 feet, whereas the growth of a person whose age is within 16..30 is 4.5..6.3 feet. The actual process of defining this fuzzy set \( \text{GROWTH} \) in FRDB-II, is illustrated in the Fig.2.

FRDB-II is capable of representing data characterizing the following types of propositions,
- *Modified propositions* - Tom is very, smart, Jim is not very much, older than Suzy etc. The emphasized words are the modifiers.
- *Composed propositions* - conjunctions or disjunctions can be used to combine two or more propositions.
**Quantified proposition** - **most** days are hot, many men and **most** women go to the gym. The emphasized words here are the quantifiers.

**Qualified proposition** - Some examples of qualified propositions are "Jim is mean" is quite true, "Jim is 400 years of age" is almost impossible.

Data can also be represented in the FRDB-II by using fuzzy relations such as Similarity and Link. The Similarity relation is reflexive and symmetrical and signifies the closeness of two attributes whereas Link is non-reflexive and non-symmetrical and is used to represent relationships between attributes. Consider a Similarity relation SPORTSMAN as shown in the table below,

<table>
<thead>
<tr>
<th>NAME</th>
<th>AGE</th>
<th>SPORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Al</td>
<td>30</td>
<td>0.6/tennis + 0.5/volley</td>
</tr>
<tr>
<td>Bob</td>
<td>35</td>
<td>0.8/tennis + 0.2/swim</td>
</tr>
<tr>
<td>Cyd</td>
<td>29</td>
<td>0.9/jog + 0.3/swim</td>
</tr>
</tbody>
</table>

Now suppose the user wishes to select Persons whose age is around 30 and they jog. In classical systems, no person would be selected. However in FRDB-II, if the user previously defines a similarity function between different sports, for example, "Jogging and Volleyball have similarity of 0.8", "Baseball and Hand-gliding have a similarity of 0.0", etc., then Al and Cyde would be selected as people of age around 30 and who jog. The modifier around has to be user predefined also. Depending on the definition of this modifier a range is created. For example, around 30 may be in the range 29.31, or 29.34, etc. Now, if Al's age is in the created range of ages and if volleyball and jogging have a reasonably high degree of similarity then Al and Cyde would be selected.

FRDB-II enables the user to express fuzzy relations via a link or strength of a relation. Consider the relation FRIENDS, with the following table,

<table>
<thead>
<tr>
<th>NAME</th>
<th>L-NAME</th>
<th>LINK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Al</td>
<td>Bob</td>
<td>0.5</td>
</tr>
<tr>
<td>Bob</td>
<td>Al</td>
<td>0.2</td>
</tr>
<tr>
<td>Al</td>
<td>Cyd</td>
<td>0.8</td>
</tr>
</tbody>
</table>

This represents a general relation that is not symmetric i.e. Al's friendship towards Bob has the link strength of 0.5 whereas Bob is friendly towards Al with the link strength of only 0.2. FRDB-II uses the concepts of links to express relations which are not symmetric. This enables the user to express fuzzy data in a relatively better way.

The FRDB-II MODIFY module allows the user to specifically pick any fact from the database for modification or it allows the user to browse through a particular class of facts in the database and then select a fact for modifications. During the process of modification the changes that are done to the initial fact are always reflected on the screen for a better understanding of the user. The system is equipped with a variety of delete options which makes the process of deleting facts very flexible. The user need not type-in a command to specify what operation has to be done. Instead, the system guides the user through a series of pop-up windows and thus forms the command for a particular task. The access of the define and modify module is a privilege of the authorized users only whereas the query module may be serviced by practically anyone. The system may generate a result of the type interval when the query operates on a fuzzy or inexact data. The Fig.3 demonstrates the execution of a query and it can be seen from the figure that the system has evaluated a value of an attribute PEN_PRICE defined as a function of another attribute COST. The generated value is of the type range.

The query language of the FRDB-II supports simple as well as compound queries. A compound query is created by using conjunctions (and) and/or disjunctions (or). In the FRDB-II system the user not only has the option to query a fuzzy set and view its results but he can also see the results in a graphical form. The system displays a graph for a fuzzy set and this feature enhances the ability of the user to better understand the results. The graph generated for a definition of a fuzzy set PLEASANT will be similar to the one presented in the Fig.1 above.

A lot of features supported by FRDB-II which are not to be seen in any of the conventional databases have been skipped in the present discussion due to the constraint on the size of the article.
CONCLUSION

In this paper we have introduced some issues related to the design of the Fuzzy Relational Database-II (FRDB-II), as an example to illustrate how it is possible to represent vague or imprecise data. Information as needed by an expert system may not always be precise and it is seen that the traditional databases are not as beneficial to Expert Systems as the FRDB-II is, which is modified to incorporate the data as required by Expert Systems themselves. However very little has been discussed about the query language of the FRDB-II due to the lack of enough room in this article for all the features supported by the FRDB-II.

REFERENCES


REASONING WITH UNIONS OF CONVEX TIME INTERVALS

Robert A. Morris and Lina Al-Khatib
Dept of Computer Science
Florida Institute of Technology
150 W. University Blvd.
Melbourne, FL 32901
(407) 786-8000 ext. 7290

ABSTRACT

Traditional interval-based representations of time assume interval convexity, i.e., the intervals are uninterrupted. This assumption makes it difficult to represent the common sense notion of a single event with "gaps". Having a representation of this species of event should contribute favorably to the ability of the human or machine to solve certain tasks, such as planning or database retrieval. This paper defines a class of discourse and knowledge object which comprises collections of convex intervals, construed as the temporal components of a single event with gaps. The formulation we choose is an extension of James Allen's interval-based approach to time representation. Allen's thirteen binary relations defined between two convex intervals is generalized to a matrix of binary relations between each pair of subintervals of two non-convex intervals.

Keywords: Knowledge Representation, Temporal Reasoning

1. INTRODUCTION

The importance of a representation of time in the formalization of common sense knowledge is indisputable. Planning, text understanding, database retrieval and task management are unthinkable without a representation of objects in time. This paper considers a class of discourse and knowledge objects which comprises collections, or unions, of time intervals. Unions of intervals are computationally useful in two ways: first, as a semantic representation of events which are interrupted in time; secondly, to represent events which recur periodically. The work of this paper is a continuation of the work of Allen on convex intervals, [1], and of Ladkin [4]. Allen's treatment of time contains a number of simplifying assumptions, an important one being the assumption of convexity of intervals, which we remove. Allowing non convex intervals into a representation results in a richer framework for a computational treatment of time, while retaining the advantages of Allen's computational methods.

2. BACKGROUND

Proceedings of the 3rd Florida Artificial Intelligence Research Symposium, 1990, by the Florida AI Research Society
c. 1990 FLAIRS 0-8620-1732-5/90/0050-0127

The study of temporal reasoning has emerged as a vital component of many areas of computer science, including artificial intelligence, software engineering, operating systems, and database design. There have been a number of approaches to the formalization of the information required to represent and reason with time.

James Allen's interval-based representation of time involves a set of thirteen primitive binary temporal relations. These consists of during (d), precedes (p), starts (s), finishes (f), overlaps (o), meets (m), equals (=), and their converses, preceded by (p), etc. A "composition table" [1] allows for the deduction of relations between arbitrary pairs of intervals x and z, based on known relations between x and another interval y, and between y and z. Allen's approach suggests a mathematical model of this calculus based on the relation algebra [4], wherein the thirteen relations make up the atoms of the algebra, and the composition table computes the result of composing the atoms together. In this model, the composition table represents a single set of reasoning with temporal entities. The general problem of reasoning with time is to compute a consistent set of relations for a set of n intervals, for arbitrary n. A related problem is to determine whether an assignment of relations between n intervals is consistent.

This problem can be formally represented as a Constraint Satisfaction Problem (CSP) [8], [9], [2]. A CSP contains a task specification consisting of a set of variables, each of which must be instantiated from a particular domain, and a set of predicates that the values of the variables must simultaneously satisfy. Restricting attention, as is appropriate here, to binary CSPs (one all of whose predicates are binary), the solution can be stated in terms of the instantiation of the first order wff:

\[ P_{1,2}(x_1, x_2) \land P_{3,2}(x_1, x_3) \land \ldots \land P_{n,1}(x_{n-1}, x_n) \]

The \( P_{ij} \) are often disjunctions of a number of "basic" predicates from a set of basic predicates.

An approach to solving CSPs consists of an implementation of the knowledge base as a constraint network, a network of nodes representing individual parameters with values from some domain, and arcs representing constraints, or relations, among nodes. Inferring is performed by propagating information throughout the network. In temporal reasoning systems, the nodes contain temporal interval values, and the arcs consist of a collection of possible binary
reasoning systems, the nodes contain temporal interval values, and the arcs consist of a collection of possible binary temporal relations between the nodes. There are at least two kinds of information which can be propagated throughout the network: node information and constraint information.

In solving CSPs, a state of network quiescence is reached, wherein no more propagation of information is required. The simplest algorithm for solving CSPs consists of generate-and-test; however, due to difficulties encountered as a result of backtracking, this approach is not generally acceptable. More efficient algorithms ([2]) are being developed for improving the computational costs of solving CSPS.

None of the attempts to formalize an interval-based temporal reasoning structure has considered intervals with gaps. There are, however, many applications in areas such as planning, task management, program verification, text understanding, and database query and retrieval, where the representation of such collections of intervals would be useful. A major obstacle seems to be the potential complexity of reasoning with such entities. The computational issues surrounding removal of the convexity assumption are beyond the scope of this paper. Future work in this area is therefore needed. This paper outlines an extension to Allen's interval-based time representation to include intervals with gaps.

Informally, whereas Allen represents intervals as convex:

\[ A: \text{[-----]} \quad \text{[-----]} \quad \text{[-----]} \]

with endpoints ts and te, a more general approach would allow for intervals with "gaps":

\[ A: \text{[---]} \quad \text{[---]} \quad \text{[---]} \]

One way of expressing convexity (uninterrupted intervals) is as follows, where x,y,u and z are intervals, "<" means "before" and "\[\]" means "inclusion":

\[ \forall x, y, z(x < y < z \land z \subseteq u \land x \subseteq u \rightarrow y \subseteq u). \]  \[11\]

Removing this constraint from the characterization of intervals results in a natural way of characterizing certain types of temporal information, as we now demonstrate.

3. EXAMPLES

By way of motivating the removal of the convexity assumption, let us consider a few contexts from natural language. There seem to be two useful classes of temporal entity which are naturally represented by intervals with gaps.

Consider the following English sentences:

1. My trip to Orlando was interrupted twice by car problems.
2. Happy hours invariably follow Friday faculty meetings.

These sentences are naturally viewed as involving references to (are made true by) collections of intervals. The first sentence suggests a collection of subintervals of a single event (i.e., a single occurrence with gaps):

\[ \text{Trip: [---]} \quad \text{[---]} \quad \text{[---]} \]

\[ \text{Car Problems: [---]} \quad \text{[---]} \]

By contrast, the second sentence suggests a collection of recurring events (viz. those described by the noun phrase "the Friday faculty meetings"). A proper characterization of recurring events is beyond the scope of this paper. Instead, we will focus on a method of representing single occurrences with gaps.

Much of the justification for an interval-based approach to temporal reasoning has assumed implicitly that the intervals are convex, and that natural language supports this contention. In fact, natural language does not seem to be so precise. It seems more accurate to say that ordinary discourse does support an interval-based semantics, but does not support the convexity assumption. In fact, convex intervals should be viewed as a special case of a class of intervals which may contain gaps.

Allen's approach can be generalized to handle intervals with gaps. The work in this area has already been commenced by Ladhkin [4]. For example, certain binary relations identified by Allen, such as BEFORE, can be illustrated as follows:

\[ A: \text{[---]} \quad \text{[---]} \]

\[ B: \text{[---]} \quad \text{[---]} \quad \text{[---]} \]

Other relations do not generalize quite as easily. For example, there is a version of STARTS (always STARTS) [4] in which one interval begins each of the intervals of the other:

\[ A: \text{[---]} \quad \text{[---]} \quad \text{[---]} \]

\[ B: \text{[---]} \quad \text{[---]} \]

E.g., "The president's address always starts the meeting". Finally, there seem to be relations which are natural in representing unions-of-convex-intervals depicting single events with gaps. For example, consider the relation INTERRUPTS:

\[ A: \text{[---]} \quad \text{[---]} \quad \text{[---]} \]

\[ B: \text{[---]} \quad \text{[---]} \]

For example, "Cats calls interrupted George's speech".

4. FORMAL REPRESENTATION

In a convex interval-based approach, an interval is represented by an ordered pair \((i, j)\) of rationals such that \(i < j\). To depict a union-of-convex-intervals, it is necessary to consider a set of their subintervals \((i, j)\) where the \(m\) th \(j\) element is less than the \(m+1\) th \(i\)th element:

\[ \{(i_m, j_m) : 1 \leq m \leq k \land (\forall n)(1 \leq n < k \rightarrow j_n < j_{n+1})\}. \]

The primitive relations are then defined on these objects.

We use the notation "ginterval" (for generalized interval) to refer to any set of convex intervals. Among the class of gintervals we distinguish those in temporal normal form. We use the notation "ginterval" (for normalized gintervals) to refer to these gintervals. Each ginterval is a union of convex intervals (which are its subintervals) separated by zero or more gaps and ordered by temporal precedence, i.e., satisfying the following condition: for each subinterval \(I\), if \(I\) is any other subinterval then \(I < J\) or \(J < I\) (\(\ldots\))
where \(<\) is temporal precedence). Allen’s convex intervals are a special case where there are zero gaps. Restricting our attention to nintervals will not result in any loss of generality since, referring to [6], we can always convexify repeatedly any set of convex intervals into temporal normal form. Convexification can be performed in \(O(n)\) time, where \(n\) is the number of subintervals.

Any ninterval can be represented by a set of pairs, each representing a subinterval with the first element the start time and the second the finish. We can use Ladin’s TUS formalism to represent a unit of time. [6]. Because nintervals have an internal structure, the quantity and variety of binary relations that can hold between pairs of nintervals increases. We distinguish between two kinds of relation that hold among nintervals, which we designate as external and internal relations. The following example illustrates each of them.

Let S1 and S2 be two nintervals defined as follows:

\[
S1 = \begin{array}{cccccc}
\text{b1} & e1 & b2 & e2 & \cdots & e3 & bn & \text{en} \\
\end{array}
\]

\[
S2 = \begin{array}{cccccc}
\text{-----} & \text{-----} & \text{-----} & \cdots & \text{-----} \\
\end{array}
\]

---time---

Informally, the external relation between S1 and S2 is the relation between the two convex intervals \(<b1, en>\) and \(<b2, em>\), as defined by Allen’s thirteen relations.

An internal binary relation is a binary relation between a convex subinterval in S1 and a convex subinterval in S2 or vice versa. In general, if S1 has \(n\) subintervals and S2 has \(m\), then there are \(2nm\) internal relations between S1 and S2.

In general, to define how two nintervals are related, we need to specify all the internal relations between their subintervals. The external binary relations can be deduced by considering all of the internal relations among the first and last subintervals of the nintervals. We call these relations boundary relations, and the corresponding intervals boundary intervals. Further, we distinguish the cross boundary relations, the relations between the first and last subintervals of two nintervals, and their converses, as well as the parallel boundary relations, those between the first pair of subintervals, and the last pair of subintervals.

The internal and external binary relations among intervals \(i\) and \(j\) with gaps can be represented as a matrix of convex relations. The rows of the matrix are the subintervals of \(i\) and the columns the subintervals of \(j\). We identify \(M_{ij}\) as the internal relation between subinterval \(k\) of \(i\) and \(l\) of \(j\), for each pair of subintervals of \(i\) and \(j\). For example, consider the following pair of intervals:

\[
I1 = \begin{array}{cccc}
\text{-----} & \text{-----} & \text{-----} & \text{-----} \\
\end{array}
\]

\[
I2 = \begin{array}{cccc}
\text{-----} & \text{-----} & \text{-----} & \text{-----} \\
\end{array}
\]

---time---

then, the matrix of internal relations between I1 and I2 is the following:

\[
M = \begin{pmatrix}
\tilde{a} & m & p \\
\tilde{b} & d & p \\
\tilde{b} & \tilde{o} & f \\
\tilde{b} & \tilde{p} & \tilde{p}
\end{pmatrix}
\]

To find the external relation from the internal relations, it suffices to examine the “parallel boundary relations” \(M_{1,1}\) and \(M_{n,m}\) (where \(n\) and \(m\) are the number of subintervals of \(i\) and \(j\), respectively), except in eight special cases. The following table is used to extract the external relations from the boundary internal relations. The eight special cases are those in which multiple entries appear. In these cases, the parallel boundary relations are not sufficient to identify the external relations between the intervals. In these cases we must also examine what we call the “cross boundary relations” in order to identify the external relation. These cases will be illustrated by examples.

In the external relation extraction table below, the rows label the internal relation \(M_{1,1}\) and the columns the \(M_{n,m}\) relation. (Again, we apply Allen’s thirteen relations defined among convex intervals).

\[
\begin{array}{cccccccccc}
\text{p} & \tilde{p} & \tilde{d} & \tilde{o} & \tilde{d} & \tilde{o} & \tilde{m} & \tilde{n} & \tilde{s} & \tilde{s} \\
\text{s} & \text{s} & \text{s} & \text{s} & \text{s} & \text{s} & \text{s} & \text{s} & \text{s} & \text{s} \\
p & \text{omp} & \text{d} & \text{odo} & \text{odom} & \text{odo} & \text{odo} & \text{odo} & \text{odo} & \text{odo} \\
\text{p} & \text{f} & \text{d} & \text{omp} & \text{odo} & \text{odom} & \text{odo} & \text{odo} & \text{odo} & \text{odo} \\
\text{d} & \text{f} & \text{o} & \text{d} & \text{odo} & \text{odom} & \text{odo} & \text{odo} & \text{odo} & \text{odo} \\
\text{f} & \text{d} & \text{o} & \text{d} & \text{odo} & \text{odom} & \text{odo} & \text{odo} & \text{odo} & \text{odo} \\
\text{d} & \text{f} & \text{o} & \text{d} & \text{odo} & \text{odom} & \text{odo} & \text{odo} & \text{odo} & \text{odo} \\
\text{f} & \text{d} & \text{o} & \text{d} & \text{odo} & \text{odom} & \text{odo} & \text{odo} & \text{odo} & \text{odo} \\
\text{f} & \text{f} & \text{f} & \text{f} & \text{f} & \text{f} & \text{f} & \text{f} & \text{f} & \text{f} \\
\end{array}
\]

Table 1. The External Relation Extraction Table. Rows represent values of \(M_{1,1}\). Columns represent values of \(M_{n,m}\).

As an illustration of this table, consider S1 and S2 above. \(M_{1,1} = \tilde{a}\), and \(M_{n,m} = \tilde{n}\). Using the external relation extraction table, the external relation between S1 and S2 is \(\tilde{a}\).

To handle the eight special cases, where the above table gives an indeterminate result for the value of the external relation, we need to consider the cross relations between S1 and S2, viz. \(M_{1,n}\) and \(M_{n,1}\). The special cases can be listed in the following diagram.

\[
\begin{array}{ccc}
\text{M(1,1)} & \text{M(n,m)} \\
\text{case 1 : } & \text{p} & \text{p} \\
\text{case 2 : } & \text{p} & \text{n} \\
\text{case 3 : } & \text{m} & \text{p} \\
\text{case 4 : } & \text{m} & \text{n} \\
\end{array}
\]
case 5 : p
case 6 : p
case 7 : m
case 8 : m

For the first four cases it suffices, in order to determine the external relation, to consider $M_{1,1}$. The following table performs the matching.

<table>
<thead>
<tr>
<th></th>
<th>p</th>
<th>b</th>
<th>d</th>
<th>o</th>
<th>m</th>
<th>m</th>
<th>s</th>
<th>s</th>
<th>f</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>p</td>
<td>b</td>
<td>d</td>
<td>o</td>
<td>m</td>
<td>m</td>
<td>s</td>
<td>s</td>
<td>f</td>
<td>f</td>
</tr>
<tr>
<td>C2</td>
<td>p</td>
<td>b</td>
<td>d</td>
<td>o</td>
<td>m</td>
<td>m</td>
<td>s</td>
<td>s</td>
<td>f</td>
<td>f</td>
</tr>
<tr>
<td>C3</td>
<td>p</td>
<td>b</td>
<td>d</td>
<td>o</td>
<td>m</td>
<td>m</td>
<td>s</td>
<td>s</td>
<td>f</td>
<td>f</td>
</tr>
<tr>
<td>C4</td>
<td>p</td>
<td>b</td>
<td>d</td>
<td>o</td>
<td>m</td>
<td>m</td>
<td>s</td>
<td>s</td>
<td>f</td>
<td>f</td>
</tr>
</tbody>
</table>

Table 2. Supplemental External Relation Extraction Table. The symbol "*" is used to designate an impossible relationship.

As an illustration, let W1 and W2 be two ngintervals defined as follows:

W1 = b1 b2 e1 e2 en
W2 = b1 b2 e1 e2 en

$M_{1,1} = m$ and $M_{n,m} = p$. Using the external relation extraction table, the external relation is either o or m. This is special case 3. By further considering $M_{1,1} = p$, and using the supplemental table above, we determine the external relation to be o.

For the last four cases it suffices to consider $M_{1,m}$. The following table performs the matching.

<table>
<thead>
<tr>
<th></th>
<th>p</th>
<th>b</th>
<th>d</th>
<th>o</th>
<th>m</th>
<th>m</th>
<th>s</th>
<th>s</th>
<th>f</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>C5</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>m</td>
<td>m</td>
<td>s</td>
<td>s</td>
<td>f</td>
<td>f</td>
</tr>
<tr>
<td>C6</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>m</td>
<td>m</td>
<td>s</td>
<td>s</td>
<td>f</td>
<td>f</td>
</tr>
<tr>
<td>C7</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>m</td>
<td>m</td>
<td>s</td>
<td>s</td>
<td>f</td>
<td>f</td>
</tr>
<tr>
<td>C8</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>m</td>
<td>m</td>
<td>s</td>
<td>s</td>
<td>f</td>
<td>f</td>
</tr>
</tbody>
</table>

Table 3. Supplemental External Relation Extraction Table

For example, let W3 and W4 be two ngintervals defined as follows:

W3 = b1 b2 e1 e2 en
W4 = b1 b2 e1 e2 en

$M_{1,1} = p$ and $M_{n,m} = m$. Using the external relation extraction table, the external relation is o or m. This is special case 6. By further considering $M_{1,m}$, which is p, and using supplemental table2, we determine the external relation to be o.

5. EXAMPLES OF BINARY RELATIONS

With complex entities it is possible to express generalizations and specifications regarding the internal relations. The most natural are those expressed in English by the adverbial modifiers "always," "mostly," and "sometimes". These modifiers can be given a natural semantics within the matrix representation of ngintervals in terms of operations on matrices. In this section, we expand on the work of Ledkin [4], who developed a formulation of the semantics of these operators. Our discussion is slightly more general than his, however. His definitions are often restricted to the case where the interest is in the binary relations between the ith subinterval of one nginterval and the ith subinterval of the other (what he calls the "matched pair" of subintervals). Our definitions are not so restricted.

An English sentence like "Happy hours invariably follow Friday faculty meetings" can be assigned a truth condition in terms of the internal relations among the subintervals of two ngintervals. On the matrix representation of the relations between ngintervals, the meaning of "always" can be defined as follows: let A and B be two ngintervals, n (m) be the number of subintervals of A (B), and R be a convex binary relation; then A always R B if, in the matrix representation of their internal relations $M_{n,m}$, each row contains exactly one occurrence of the relation R. (In symbols: $W_i : 1 \leq i \leq n, W_j : 1 \leq j \leq m : M_{ij} = R$).

In the special case where $n=m$, the meaning of "always" can be defined in terms of an operation which compares the diagonal of a matrix of relations. For example, the sentence "I always r's j" where r is one of Allen's relations if and only if the matrix of relations between i and j is as follows:

<table>
<thead>
<tr>
<th>r</th>
<th>r</th>
<th>r</th>
</tr>
</thead>
</table>

For example, consider the sentence "The trip was always interrupted by car problems". This sentence can be naturally viewed as involving the convex relation "meet" between subintervals of two ngintervals consisting of the trip and the car problems. More specifically, this sentence can be made true by the following pair of ngintervals:

Trip: b1 b2 e1 e2 en
Car Problems: b1 b2 e1 e2 en

In matrix form:

<table>
<thead>
<tr>
<th>p</th>
<th>p</th>
<th>p</th>
<th>p</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>m</td>
<td>m</td>
<td>p</td>
<td>p</td>
<td>p</td>
</tr>
<tr>
<td>p</td>
<td>p</td>
<td>m</td>
<td>m</td>
<td>m</td>
</tr>
</tbody>
</table>

The operators "sometimes" and "mostly" can be treated similarly. Informally, "mostly" can be defined as follows: let A and B be two ngintervals, n (m) be the number of subintervals of A (B), and R be a convex binary relation;
then A mostly R B if, in the matrix representation of their internal relations $M_{n,m}$, the ratio of the number of rows which contains R to the total number of rows is greater than .5. For example, the sentence "My trip was mostly interrupted by car problems" can be made true by the following pairs of nintervals:

<table>
<thead>
<tr>
<th>Trip:</th>
<th>-----</th>
<th>----</th>
<th>------</th>
<th>----</th>
</tr>
</thead>
<tbody>
<tr>
<td>Car</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Problems:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In matrix form:

$$
\begin{bmatrix}
  m & p & p & p \\
  \bar{p} & \bar{m} & p & p \\
  p & \bar{p} & m & p \\
  \bar{p} & \bar{p} & \bar{p} & m
\end{bmatrix}
$$

The ratio of the number of rows containing "meets" to the total number of rows (4) is .75, which satisfies the truth definition of sentences involving the operator "mostly".

Finally, "sometimes" can be given an meaning whereby A sometimes R B if there is a row of the matrix representation of the internal relations between A and B that contains the value R. For example, the English sentence "Telephone calls occasionally (sometimes) interrupted the meeting", can be made true by the following pair of nintervals:

<table>
<thead>
<tr>
<th>Meeting:</th>
<th>-----</th>
<th>------</th>
<th>------</th>
<th>----</th>
</tr>
</thead>
<tbody>
<tr>
<td>Calls:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This section has illustrated how the matrix representation of nintervals can be useful in providing a semantics for temporal adverbial modifiers such as "always", "mostly" and "sometimes". This analysis will be expanded in future work to include an extension to the notion of composition to include these operators.

6. COMPOSING NON-CONVEX INTERVALS

The relaxation of the semantical constraints of interval models by eliminating the convexity condition changes the algebraic representation of the calculus. The representation of unions-of-intervals as sets of their subintervals increases the "internal structure" of the objects. Intuitively, the result of this increased complexity is a proportional increase in the complexity of the relations that may exist between these objects. In our matrix representation, the rows and columns of a composition table for nintervals would comprise matrices of possible convex relations between pairs of intervals. Clearly, because the number of subintervals of an ninterval can be arbitrarily large, the number of matrices of binary relations for every pair of nintervals is infinite. The size of such a table is clearly too large for any practical automated system. In the preceding section, we suggested that for many natural contexts, we are interested in a restricted subclass of matrices (e.g., square matrices), and a restricted subset of the matrix (e.g., the diagonal of a square matrix when interpreting adverbial operators). The sorts of matrices of interest will depend in general on the application. In this section, we will sketch a general method for defining a composition operator for nintervals using a matrix representation of internal relations between pairs of nintervals.

The composition method we propose is a simple generalization of the method of composition proposed by Allen for convex intervals. Given two interval matrices $P^{n,m}$ and $R^{m,k}$ of dimensions $n \times m$ and $m \times k$ respectively, represented as follows:

$$
P = \begin{bmatrix}
P_{1,1} & P_{1,2} & P_{1,3} & \cdots & P_{1,m} \\
P_{2,1} & P_{2,2} & P_{2,3} & \cdots & P_{2,m} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
P_{n,1} & P_{n,2} & P_{n,3} & \cdots & P_{n,m}
\end{bmatrix}
$$

$$
R = \begin{bmatrix}
R_{1,1} & R_{1,2} & R_{1,3} & \cdots & R_{1,k} \\
R_{2,1} & R_{2,2} & R_{2,3} & \cdots & R_{2,k} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
R_{m,1} & R_{m,2} & R_{m,3} & \cdots & R_{m,k}
\end{bmatrix}
$$

The composition operation results in a matrix $Q = P \circ R$ comprising the following:

$$
Q = P \circ R = \begin{bmatrix}
Q_{1,1} & Q_{1,2} & Q_{1,3} & \cdots & Q_{1,k} \\
Q_{2,1} & Q_{2,2} & Q_{2,3} & \cdots & Q_{2,k} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
Q_{n,1} & Q_{n,2} & Q_{n,3} & \cdots & Q_{n,k}
\end{bmatrix}
$$

where $Q_{i,j} = \bigcap_{k=1}^{m} P_{i,k} \circ R_{k,j}$. This is just the operation defined in the binary constraint networks of [6] for solving CSPs (see also [10]). For example, consider the intervals:

i: | ----- | ----- | ---- | ---- | ------ |
|-----|-----|-----|-----|-------|

j: | ----- | ---- | ------ | ---- | ---- |
|-----|-----|-----|-----|-------|

k: | ---- | ----- | ------ | ---- | ---- |
|-----|-----|-----|-----|-------|

The matrix representations of the internal relations between i and j (call it P) and between j and k (call it R) are as follows:

$$
P = \begin{bmatrix}
s & p & p & p & p \\
\bar{p} & \bar{s} & p & p & p \\
\bar{p} & \bar{p} & \bar{p} & \bar{p} & f \\
\bar{p} & \bar{p} & \bar{p} & \bar{p} & f
\end{bmatrix}
$$

$$
R = \begin{bmatrix}
o & p & p \\
\bar{o} & m & p \\
\bar{p} & \bar{p} & f \\
\bar{p} & \bar{p} & \bar{p}
\end{bmatrix}
$$

Therefore, the composition matrix, $P \circ R$, expressing the internal relations between i and k is:
\[ Q = \begin{array}{ccc}
\text{pom} & p & p \\
\text{d} & o & p \\
\text{p} & \text{pom} & p \\
\text{p} & \text{pom} & f \\
\end{array} \]

7. SUMMARY AND FUTURE DIRECTIONS
In this paper, we have explored relations which can express truths about collections of intervals, including both intervals representing events with gaps, as well as intervals representing recurring events. In future endeavours we seek an algebraic model for representing a useful subclass of these relations, as well as an exploration of the relative efficiency of computational methods for computing the consistency of a network of intervals with gaps. We also seek a more extensive treatment of composition for relations involve temporal operators such as “always” and “mostly”.

8. ACKNOWLEDGEMENT
The authors wish to thank Peter Ladkin for the valuable suggestions and criticisms of earlier versions of this paper.

9. REFERENCES


Analysis of General Tabular Parsing  
For Natural Language Translation

FRANK HADLOCK
Computer Science Department
Tennessee Technological University
Cookeville, TN
and
Mark Fishman
Computer Science Department
Eckerd College
St. Petersburg, Fl.

1 Problem Description and Previous Results

The problem addressed in this effort is an extension of that described in [5]. The basic objective is to translate natural language functional specifications for large, embedded real-time distributed software systems into a target language which models system behavior and domain knowledge contained in the specifications. The translation should capture the information content of the specifications to a level of detail sufficient for such applications as checking consistency and completeness of specifications, generating data flow diagrams, etc. Previous results have included the development of a target language referred to as the State Event Calculus (or State Event Calculus model) as well as the development of a translator based on the General Tabular Parsing algorithm [5]. The grammar and language developed previously were intended to demonstrate feasibility of the approach; they are limited to modeling system behavior which can be described by simple stimulus-response format. The translator is capable of handling this sentence format but has no mechanism for extension. In order to achieve further progress with respect to the basic problem, the following related subproblems need to be addressed:

1) extend the SeCalc meta model for software system behavior in order to model more complex system behavior as well domain knowledge, and
2) extend the SeCalc translator to permit sublanguage extension through interactive semantic editing of the grammar, and
3) develop techniques for inferring semantic actions associated with rules edited through extension process.

2 The Target Language: The State Event Calculus Meta Model with supporting ‘World Model’

State and event frames, along with temporal/causal relations make up the SeCalc meta model developed previously [5]. We extend the model here by extending the basic frame format, as well as the inventory of frame types.

The State Event Calculus meta model, hereafter referred to as SeCalc, models system behavior and domain knowledge using the following frame types: Event frames, State frames, Entity frames, and Script frames. Loosely, event and state frames provide for the description of system behavior, along with script frames, which provide for the description of system processes in terms of simpler processes, or the description of complex conditions. Entity frames provide for the description of system and domain objects. Re-
lations between actions and entities, between actions, and between entities are described by frames through relation slots.

**Event Frame:** Perhaps the most central frame type is the event frame, which is motivated by the case grammar approach to representing the meaning of a simple sentence or independent clause. The idea is that the action is central to the sentence/clause meaning, with the case relations between the action (verb) phrase and the entity (noun/prepositional) phrases completing the meaning. The case relations given in the next section, along with system entities and actions, are used to form Event Frames, a mechanism for representing system actions of limited duration. The format of an Event frame provides slots for Action, Case Roles, and Frame Relations. A Case Role entry is an entity related by Role to the entry in the Action slot while a Frame Relation entry indicates another state or event related to the current event temporally or causally, etc.

**State Frame:** States of system entities involve one of the three basic state relations: is in some state, has some attribute/property, is related to a second entity through some standard entity relationship. A primary entity, state relation, and state, attribute or second entity, make up a state frame. The format of a State frame provides a slot for a Primary Entity and a slot for a State Relation. Depending on the State Relation, there is a third slot for State, Property/Attribute, or Secondary Entity. Thus a state can model situations such as: buffer is full, or checksum has even parity, and X value is greater than Y value. As with Event Frames, State Frames may contain Frame Relation slots to model causal relations between the state and events and other states.

**Entity Frame:** System and domain entities are modeled by an Entity Frame. The format of an Entity Frame provides a slot for the entity, as well as its type. If the type is module (system), then the Frame will provide Function Slots determined by action - object slot entries of Event Frames in which this module appears as Agent, along with other slot types which would serve to characterize a system module (Input, Output, Calls, Called-By, etc.). If the type is data, then the Frame will provide Source-Destination Slots determined by the Agent-Destination slots of a Send Event of which this data was the object, or the Source-Agent slots of a Receive Event of which this data was the object.

**Script Frame:** The format of a Script Frame provides a mechanism either for representing a complex antecedent condition, along with a consequent event or state triggered by the condition, or for representing a complex event or system process in terms of a network of simpler states and events. Slots occur in pairs, each pair consisting of a Condition slot and a consequent Entry or Effect slot (depending on whether the slot filler is an event or a state). The Condition slot entry is a Boolean expression with State and Event Frame IDs as operands: an Event Frame ID has value true or false depending on whether the event occurs or does not; a State Frame ID has value true or false depending on whether the state relation holds or not. If the Condition expression has value true, then the event or state indicated by the Consequent Slot is triggered. In this way, a Condition Frame models a complex condition needed to trigger an event or state. In case of a series of Antecedent- Consequent slot pairs, it is assumed that at most one of the Antecedent expressions is true at any time. Thus the script frame provides a mechanism for indicating entry points into a network of events, states and conditions.

### 3 Source Sublanguage Model

In order to translate from natural language functional specifications to a SeCalc frame language representation, we must model the source sublanguage at various syntactic and conceptual levels.

#### 3.1 Word Classes

Words are classified according to their correspondence to components of the SeCalc model. No distinction is made between a verb and a nominalization of the verb (e.g. receive and receipt). The basic word classes are listed below, along with standard syntactic classification, and the corresponding SeCalc components.

<table>
<thead>
<tr>
<th>Standard Class</th>
<th>Sub-Classification</th>
<th>Functional Correspondence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discard</td>
<td>Article</td>
<td>No correspondence to any model component.</td>
</tr>
<tr>
<td></td>
<td>Aux-Vrb</td>
<td></td>
</tr>
<tr>
<td>Entity</td>
<td>Noun</td>
<td>Event/state/entity frame slot entry.</td>
</tr>
<tr>
<td>Action</td>
<td>Transitive</td>
<td>Event frame action slot entry.</td>
</tr>
<tr>
<td></td>
<td>Verb Form</td>
<td></td>
</tr>
<tr>
<td>State</td>
<td>Intransitive</td>
<td>State Relation</td>
</tr>
<tr>
<td></td>
<td>Verb</td>
<td></td>
</tr>
<tr>
<td>Modifier</td>
<td>Adjective</td>
<td>Slot co-entry with entity word.</td>
</tr>
<tr>
<td></td>
<td>Adverb</td>
<td></td>
</tr>
</tbody>
</table>

134
with action word.

Pointer Pronoun Replaced by referenced entity in slot pointer would have occupied.

Relation Conjunction/ Temporal/causal Subordinator relation between events.

Preposition Case role relation indicating event role slot.

The main word classes, action, entity, and relation are further classified as follows.

System actions are classified in two ways: how they relate to system or domain entities (case role classification), and how they relate to other system or domain actions (coherence relations).

3.1.1 Action-Entity or Case Relations:

From processing functional specifications and analyzing action-entity relations, the following list of action-entity relations is proposed as a complete list of standard case relations for system behavior. Domain knowledge may require a somewhat different list.

List of action-entity relations for distributed real-time systems

Agent, Object, Temporal, Multiple-Object, Compound-Object, Beneficiary, Instrument Source, Destination, Result, Location

These are standard action-entity, or case, relations except for Multiple-Object, Compound-Object, and Result which occur in describing system behavior as follows. Multiple-Object refers to the relation between the second operand and a processing action; new value and average would play Object and Multiple-Object roles respectively in ... the new value will be compared with the average. The compound-object relation is introduced in order to relate a notify or verify action to the abstract entity (either an event or a state) serving as it's object; the state :parity is even, is the compound-object of the verify action in ... will check to see if the parity is even .... Result is the relation between a processing or verify action and the entity produced by the action; the implicit Boolean result produced by the verify action plays the Result role in ... will check to see if the parity is even ....

Case role classification refers to the type of roles which are required or are optional to complete the meaning of the clause containing the action word. The Agent role is always required but is often omitted since it can be inferred from context; Object is required unless some other form of object (Multiple Object or Compound Object) is required. The table below lists the different action classes in the target language, omitting Agent and Object (unless some other form is required). Subclassification is made by considering the entity classes of the entities playing the required case roles while further classification is performed based on action-action, or coherence relations discussed in the next section.

3.1.2 Case classification of actions:

Action - Case Role Table

<table>
<thead>
<tr>
<th>Action Classes</th>
<th>Required Roles</th>
<th>Optional Roles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Send</td>
<td>Destination</td>
<td>Instrument</td>
</tr>
<tr>
<td>Receive</td>
<td>Source</td>
<td>Instrument</td>
</tr>
<tr>
<td>Store</td>
<td>Destination</td>
<td>Location</td>
</tr>
<tr>
<td>Retrieve</td>
<td>Source</td>
<td>Location</td>
</tr>
<tr>
<td>Display</td>
<td>Destination,</td>
<td>Destination</td>
</tr>
<tr>
<td>Notify</td>
<td>Compound Object</td>
<td></td>
</tr>
<tr>
<td>Control</td>
<td>New State</td>
<td>Instrument</td>
</tr>
<tr>
<td>Transfer</td>
<td>Source, Destination</td>
<td>Boolean Result</td>
</tr>
<tr>
<td>Verify</td>
<td>Compound Object</td>
<td></td>
</tr>
<tr>
<td>Monitor</td>
<td>Result</td>
<td>Multiple Object</td>
</tr>
<tr>
<td>Process</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3.1.3 Coherence classification of actions:

Coherence relations are those relations between system actions. For modeling system behavior, the following two axioms, as part of the supporting "world model" for SeCalc, are identified, from which certain coherence relations are implied.

Conservation of Data: The only points at which data entities can originate are with external device entities, and as a result of a processing action, and

Rational Behavior: When data is received, is retrieved, or is produced by processing, then a meaningful operation must be performed to utilize it.

As a result of these axioms, we define a coherent action sequence (within a module) as one which is initiated by a receive or retrieve, or a control event or
3.2 An Attributed Context Free Grammar

An attributed context free grammar is used to approximately model the source language. By approximately model is meant that the language defined by the base grammar (i.e., ignoring attributes) is a superset of the source language; while that defined by the attributed grammar with attribute constraints is a closer approximation. This approach allows for a compact base grammar; at present the base grammar consists of thirty some rules. The approach employs what is referred to as a Semantic grammar which means that syntactic types are chosen to correspond to conceptual entities in the target language; for the State Event Calculus as target language, this means that source language syntactic types are chosen to correspond to module specification, coherent-action sequence, event, state, event-event relation, action-entity relation, action, entity or case phrase lists, entity, etc.

In order to establish a small set of points at which the SeCalc translator can interact with the user, in case of unresolvable ambiguity, anaphora or ellipsis, the grammar is divided into rule packets as described below.

Packet Structure of Grammar

Packet 1
- deals with structure of module specification and coherent action sequences.

Packet 2
- deals with structure of events and states in terms of action phrases and case phrase lists.

Packet 3
- defines case phrase lists as a list of case phrases, and a case phrase in terms of a case relation word or a article discard word and the entity.

Packet 4
- deals with the structure of verb phrases in terms of auxiliary and action phrases.

Sample rules are given from each packet below.

Sample Rules
The attributes associated with the characters carry
class information and text pointers. Attribute con-
straint rules provide a mechanism for word sense dis-
ambiguation as well as a mechanism for generating the
frame structures in the target SeCalc language.

4 SeCalc Translator

The components of the SeCalc Translator are the Gen-
eral Tabular Parsing algorithm, described in [5] and
restructured to parse with respect to packets $P_i, i = 4$
down to 1, along with mechanisms for ensuring that
attribute constraints are observed and for evaluating
attributes.

4.1 The General Tabular Parsing Al-
gorithm

The SeCalc Translator is based on the General Tabu-
lar Parsing Algorithm, GTP, described below. GTP
was developed as an extension to the CKY algorithm
[7],[9]; like the extension due to Graham, Harrison and
Russo [3], it employs the same tabular data structure,
entries may be either primitive or phrase items (de-
\text{fined below) so that, unlike the CKY algorithm, nei-
ther GTP nor the Graham,Russo Harrison extension
require that the grammar be in Chomsky normal form.
At the same time, the General Tabular Parsing algo-
rithm inherits the bottom up approach, is inherently
parallel, and is independent of sentence structure out-
side the local phrase. Consequently it degrades grace-
fully in ungrammatical situations, parsing all phrases
which are syntactically correct in themselves, regard-
less of errors external to the phrase. This means that
translation can often be accomplished even when there
are syntax errors (or grammar deficiencies), and that
user interaction can be limited when translation can-
not be accomplished. Since the table can store multi-
ple entries, ambiguity can be accommodated. A descrip-
tion of GTP follows.

General Tabular Parsing algorithm:

central data structure is a tabular structure $T_{i,j}$
with

- \text{primitive & phrase item entries}
  - Primitive Item : $X$, where $X$
is terminal/nonterminal
  - Phrase Item : $\alpha \beta$ where
    $\alpha \beta$ = RHS some rule
    $\text{Item} \in T_{i,j} \Leftrightarrow \text{component}$
to the left of the dot
    ($X$ or $\alpha$) generates the sentence
    portion beginning with the $j$th word
    and consisting of $i$ words.

- \text{Processing Primitives}:
  - $j$th sentence word $w_j$ is entered as
  - Primitive Item $w_j$, in $T_{i,j}$
  - When $X$ is entered in $T_{i,j}$, it is processed:
    - If $Y \rightarrow X$ is rule, then enter $Y$. in $T_{i,j}$
    - If $Y \rightarrow X \beta$ is rule, then enter $X \beta$ in $T_{i,j}$

- \text{Processing Phrases}:
  - When $\alpha.X \beta \in T_{i,j}$, it is processed:
    - Search column $j+i$ for $X$
    - If $X \in T'_{i,j}$ then enter $\alpha X \beta$ in $T_{i+j',j}$
  - When $\alpha.X \in T_{i,j}$, it is processed:
    - Search column $j+i$ for $X$
    - If $X \in T'_{i,j}$ then enter $\alpha X$ in $T_{i+j',j}$

4.2 SeCalc Translation Process

Besides the parsing, the translator performs the fol-
lowing tasks:

\begin{itemize}
  \item checking of attribute constraints and
  \item attribute evaluation.
\end{itemize}

Initially, attributes are obtained from dictionary
entries, text pointers indicating individual words.

As phrase items are extended, attribute values are
checked in order to verify that the primitive
item entry is a valid extension.
When a phrase item is completed, the translator computes the attribute values of the primitive item corresponding to the Left Hand Side.

4.3 Error and Ambiguity Handling

In case of grammar deficiencies, or ungrammatical constructions, or in the case of ambiguity, the translator will revert to interactive mode. The user will be asked to respond to various interpretations due to the problem situation: too many plausible interpretations or none, arising from ambiguity, anaphora or ellipsis.

Intermediate interpretations or parses are constructed routinely from General Tabular Parse table at the end of parsing with respect to a rule packet. Not only does the packetization allow the use of case information inferred from parsing with respect to the action packet, but it provides a small number of points during the parse for evaluating the situation and interacting with the user if necessary before proceeding. A partial parse $p_i$, constructed while parsing with respect to the $i$th packet, is a string $p_i = X_1 \ldots X_n$ over $\Sigma \cup N$ such that, for some segmentation $s_1 \ldots s_n$ of a previous parse $p_{i+1}$ ($p_i$ is necessarily the source sentence $s$), $X_j \Rightarrow^* s_j$, $j = 1 \ldots n$. Note that there is no guarantee that $S \Rightarrow^* p_i$; if it does, then General Tabular Parser will find the derivation as the remainder of a derivation of the source sentence. A parse $p_i$ will be called $C$-viable if it satisfies some necessary criterion $C$ that $S \Rightarrow^* p_i$. One such criteria would be that some precedence relation $\tau \in \{\leq, =, \geq\}$ exist between $X_j$ and $X_{j+1}$ for each $j$. A stronger condition can be expressed in terms of the extended precedence relations described in [4], which associate rule index sets with relations, and the rule indices occurring in $P_i$...$P_{i+1}$. It is easily seen that the tabular nature of General Tabular Parsing supports the generation of these parses. The characters $X_j$ of a partial parse $p_i$ are formed from source characters which will be parsed during some later packet and primitive items produced during the parse with respect to the $i$th packet.

Let $P_i$ be the set of viable parses. If $P_i$ is empty or has more than one viable parse, the translator will revert to interactive mode. In case of no viable parses, the tabular nature of the parser supports the following error recovery paradigm. By approximate phrase will be meant a string $X_1 X_2 \ldots X_k$ over $\Sigma \cup N$ in which each character $Y_i$ has a optional edit operation: $+, -, or \rightarrow$, representing insertion, deletion, or replacement by another character, the result being a string $X_1 X_2 \ldots X_k$ occurring as the right hand side of a grammar rule $A \rightarrow X_1 X_2 \ldots X_k$. For example, consider the phrase "access" from the sentence "An authorization code will be issued for access to the database." The primitive item in the table would be $\langle \text{Action word} \rangle$. The right hand side of the rule $\langle \text{Action} \rangle \rightarrow \langle \text{Discard list} \rangle \langle \text{Action word} \rangle$ can be formed by $\langle \text{Discard list} \rangle \langle \text{action} \rangle \langle \text{Action word} \rangle$ from consecutive table entries. A cost is assigned to an approximate phrase item based on the number and nature of edit operations to produce the approximate phrase item. An entry of the left hand side of the rule as approximate primitive item is made only if the cost does not exceed some threshold; in this case the approximate primitive item is assigned the same cost and entered into the table. Viable parses are assigned costs in additive fashion, with exact primitive items and source characters having zero cost, and these are then presented to the user in order of increasing cost until an acceptable one is found. An example is shown below:

1. An authorization code $^2$ will be issued
2. for $^3$ access $^5$ to the database

1. $\langle \text{Entity} \rangle \langle \text{object} \rangle \langle \text{Action} \rangle \langle \text{Give} \rangle \langle \text{Relation} \rangle \langle \text{Event} \rangle \langle \text{Enable} \rangle$
2. $\langle \text{Entity} \rangle \langle \text{Str/Rtr} \rangle \langle \text{5(Entity) Destination} \rangle$

User interaction for ambiguous situations follows a similar format, except there is no ranking of viable parses. When possible, multiple viable parses are presented simultaneously, as shown below:

1. The rate module $^2$ will determine
2. premature beats $^3$ with the average R-R interval

1. $\langle \text{Entity} \rangle \langle \text{gen} \rangle \langle \text{Action} \rangle \langle \text{process} \rangle$
2. $\langle \text{Entity} \rangle \langle \text{Object} \rangle$
3. $\langle \text{2(Entity) Instrument} \rangle \langle \text{3(Entity) Specification} \rangle$

Presumably the user would select 4 by mouse or keypad, since 5 would indicate that premature beats having an average R-R interval are being determined, a contradiction. It should be noted that it would require considerable domain knowledge, along with an inference mechanism, for the system to resolve the ambiguity. The SeCalc model provides mechanisms for storing this domain knowledge, since the SeCalc model for some patient monitoring system would acquire entity frames for "heart beat" and "R-R interval", along with relation slots indicating that a "heart beat" has an "R-R interval", and attribute slots indicating that a "heart beat" has a time attribute whose value can be premature, normal, or delayed, and that an "R-R
interval" has a length attribute whose value can be short, average or long. However disambiguating the sample sentence requires reasoning that a premature beat cannot have an average R-R interval, and some mechanism to trigger this reasoning process.

References


[3] GRAHAM, S., HARRISON, M., RUSSE, W., An Improved Context-Free Recogniser,


[4] HADLOCK, F. Derived Grammars with Application To Syntactic Analysis,

Third Annual Princeton Conference on Information Science, 1969

[5] HADLOCK, F., FISHMAN, M., ANGER, F., and RODRIGUEZ, R.,

Inference of State-Event Calculus Model for Software System Behavior from Natural Language Specifications,


[6] HARRIS, M.D., Natural Language Processing, Reston, 1985

[7] KASAMI, T., An efficient recognition and syntax-analysis algorithm for context free languages,


[8] KNUTH, D.E., On the translation of languages from left to right,

Information and Control, 8, pp 607-639, 1965

[9] YOUNGER, D.H., Recognition and parsing of context-free languages in time \( n^3 \),

Information and Control, 10, pp. 196-208, 1967
Use of Fuzzy Relational Information Retrieval Techniques for Generating Control Strategies in Resolution-Based Automated Reasoning

Ladislav J. KOHOUT
and
Yong-Gi KIM

The Center for Expert Systems and Robotics
Department of Computer Science
Florida State University
Tallahassee, Florida 32306, USA.

2 The use of strategies in automated reasoning schemes

Current automated theorem prover ITP adopts strategies called set of support strategy and weighting strategy. The reason for using strategies is that the automated reasoning program can avoid many fruitless paths by their judicious and "informed" application. Without a suitable strategy guiding the inference, too many often irrelevant clauses may derive, and those clauses may lead the program easily into a blind alley. Therefore, the strategies are the must in any serious use of automated reasoning. The set of support strategy is one of the most powerful restriction strategies in the resolution-based automated inference systems. The set of support strategy fords a reasoning program from applying an inference rule unless at least one of the potential parents to which it is being applied has been deduced from some specified subset of the input clauses. Even though the set of support strategy eliminates many fruitless clauses from the inference stream, it is often not powerful enough to produce the conclusion in acceptable time. Hence, weighting strategy is being used with the set of support strategy, in the current theorem prover ITP in addition to the set of support. The weighting strategy assigns some priorities to each term, literal, and clause. With the weighting, one can assist the reasoning program by contributing some of one's own experience capturing one's intuition, in order to give the program hints. Weighting means assigning "weights" to various concepts. The lighter the weight is, the sooner the program will look at the clause. Unfortunately, this weighting strategy is too heuristic and too dependable on the subjective side of one's experience or intuition.

Here, we propose to apply the fuzzy information retrieval technique [5] instead of the weighting strategy. The information retrieval technique applied to the automated theorem prover might be capable of removing the deficiencies of the weighting strategy. The information retrieval technique of the new scheme may select the weighting patterns and weights automatically, replacing the manual selections done by users in the ITP.
3 The global activity of the ITP

In order to elucidate the new scheme further, we have to have a closer look at the global activity of ITP. The user enters theorems in the clause form into the ITP. These clauses are distributed between a list of axioms list, the set of support list, the have-been-given list, and the demodulator list. The fundamental operation consists of the following steps:

1. Choose a clause from the set of support list. Call this clause "the given clause".

2. Infer a set of clauses that have the given clause as one premise; as other premises the clauses are selected from the axioms list, the have-been-given list, and the demodulator list, depending on the chosen type of inference process.

3. For each generated clause, "process" it (i.e., simplify it, perform subsumption checks, etc.).

4. Move the given clause from the set of support list to the have-been-given list.

The operation of the ITP consists of repeated execution of these four steps until either the set of support list has become exhausted or a contradiction has been found. The user enters selections (i.e., which inference rule should be used and how the given clause is picked up) into the ITP with the clauses of the axioms defining the problem. The user-controlled options govern the step 1, 2, and 3 above. These user-controlled options include selection of the inference rules (e.g., binary resolution, unit resolution, and hyperresolution, etc.) and the weighting scheme for each term, literal, and clause. Since too many clauses are generated through the repeated steps, specific weights are assigned to each term, literal. The clause to be picked up first is the one with the lowest weight.

4 Replacement of weighting strategy by a Fuzzy Information Retrieval scheme

The priority of the second premiss, in the activity step 2 of the previous section, is determined by the weight heuristically assigned by the user of the ITP. We replace the weighting strategy by the Fuzzy Information Retrieval (FIR) scheme (Kohout and Bandler [5]), thus making the value of the assigned priority the function of fuzzy logical request and fuzzy relational request (Kohout and Bandler [6]) of FIR. The major advantage of our new scheme is the fact that the order-like relations determining the priority of the clauses selected to be entered into the inferential stream of the ITP can be identified from the experimental data by fast fuzzy relational algorithms (Bandier and Kohout [4]).

The functional specification of the activity of FIR used to select the relevant clause is as follows (Kohout and Bandler [5]):

What is involved in fuzzy Information Retrieval of clauses can be expressed essentially by means of the following four items:

1. A set D of clauses d.

2. A set T of descriptors t (for example properties of clauses).

3. A clause-descriptor relation R, which is a fuzzy relation such that

\[ R_{RF}(D \rightarrow T) \]

Then \( R_{d,t} \) is the degree to which clause \( d \) is related to descriptor \( t \), which can be viewed as the degree of relevance of the features described by descriptor \( t \) to clause \( d \).

4. A set S of search requests \( s \) (fundamentally, the lattice generated by \( T \) with the operators and, or, not.

Fuzzy search requests are elements of set of syntactically well-formed expressions, such as

\[(t_1 \land t_2) \lor (t_3 \land t_4)\]

They are dealt with in the following way.

1. Simplified by use of the thesaurus.

2. Satisfied by use of relation \( R \) which provides as answer a fuzzy subset \( A \) of clauses \( D \).

3. This is then reordered by decreasing rank of the fuzzy weights, and printed out down to a cut-off criterion (that is, the \( \alpha \)-cut of \( A \) is listed).

4. The floating \( \alpha \)-cut is defined by the distance \( \delta \) from the top value of the reordered list. This introduces content-adaptive filtration in the source - consumer interaction. This filtration is also used extensively for generating questioning strategy in the inferential scheme of CLINAID [7].

The appropriate characteristics of descriptors have to be determined empirically, by a series of carefully designed experiments. It is clear that the set of relevant properties of the elements involved in these experiments is strongly dependent on the mathematical characteristics of the problems presented to the ITP.
5 Experimentation with various fuzzy descriptors

The way the relation between the clauses and their set of descriptors is formed, may have a substantial impact on the performance of the employed inferential strategies. The purpose of the matrix is to capture essential characteristics of the axiomatic base defining the theory within which we attempt to prove some theorems that interest us. A concrete matrix build for a certain axiomatic theory is not universal, but strongly depends on the set of goals set for the inferential process.

It is not possible to present the full description of the experiments and the results here, due to space limitation, so we present an illuminating comparison of various strategies on an example from the Group theory. This demonstrates some advantages of using fuzzy relational product for forming the structures selecting the appropriate clauses during the inferential process.

The commutator theorem, "In a group, if square of every element is the identity, the group is commutative", was chosen to illustrate the effect of various inferential strategies on the length of the proof. The set of clauses/axioms listed below was adopted from [2].

axioms:

1) P(e,x,x); left identity
2) P(x,e,x); right identity
3) P(g(x),x,e); left inverse
4) P(x,g(x),e); right inverse
5) P(x,y,u) \rightarrow P(y,x,v) \rightarrow P(u,z,w); \text{ associativity}
6) P(x,y,u) \rightarrow P(y,x,v) \rightarrow P(x,v,w); \text{ associativity}
7) P(x,e,x); square of every element is identity

set of support:

8) P(a,b,c); proof by contradiction. two elements exist
9) P(b,a,c); which do not commute.

Where the predicate "P(x,y,z)" expresses the fact that the product of x and y is z is the group operation.

An ITF system installed on VAX11-780 was used to perform all our experiments. Below, we present a comparison of results employing different strategies in proving the commutator theorem of the Group theory.

Strategy 1:
Employing the binary resolution with factoring, the system could not deduce the empty clause within 2 hours and the process was aborted. Theoretically, the binary resolution with factoring should produce the empty clause.

Strategy 2:
Employing hyperresolution only, without weighting pattern, the system produced the empty clause in 120 steps.

Strategy 3:
Employing hyperresolution and UR-resolution simultaneously without weighting pattern, the system produced the empty clause in 73 steps.

Strategy 4:
Employing hyperresolution and UR-resolution simultaneously with weighting pattern c:+3, b:+1, a:+1, the system produced the empty clause in 53 steps. To get the weighting pattern which can produce this improved result, many experiments had to be executed, in order to find the appropriate heuristic pattern.

Strategy 5:
Employing hyperresolution and UR-resolution simultaneously with weighting. A weighting pattern automatically extracted from the fuzzy clause-property relation described above, produced the proof in 39 steps. A fast fuzzy relational algorithm based on the computation of fuzzy relational closures using a triangle fuzzy product with Lukasiewicz implication operator was used to extract the local preorder which determined the weighting pattern employed.

The set of objects of the fuzzy matrix used in Strategy 5, was formed by the axioms and the immediate consequences of the axioms, that were unified with the set of support. The set of properties was formed by the constants a,b,c,g(a),g(b), g(c),and by g(e).

The degrees of relatedness between the clauses and the properties used to form a fuzzy matrix were determined by application of the following rules:

1. If the property j is an element of the clause i, then the degree of the relatedness $D_{ij}$ is 1.

2. If the property j is not an element of the clause i, then the degree of the relatedness $D_{ij}$ is 0.

3. If the property j is an element of a subterm (i.e., g(a)) of the clause i, then the degree of the relatedness $D_{ij}$ is 0.5.

4. If the property j is an element of a subterm of a subterm (i.e., g(g(a))) of the clause i, then the degree of the relatedness $D_{ij}$ is 0.5 x 0.5, and so on.

5. If the property j is an element of the clause i, and the property j is an element of a subterm of the clause i at the same time (i.e., P(a,b,g(a))), then the degree of the relatedness $D_{ij}$ will be bigger one applied to the case using
Although interesting results can be obtained by offline computations by which appropriate preorders are formed to determine the weighting pattern that is consequently put into the ITP manually, systematic experimentation requires a better automatic support. Effective experiments with the fuzzy relations generating inferential strategies can be performed systematically only within the framework of IFP described above. This in turn requires an appropriate conceptual framework and methodology.

A brief programmatic outline is given below.

6 A conceptual framework for capturing the heuristics of the strategies of inference

The relational architecture of the Fuzzy Information Retrieval Scheme and the inferential processes performed by the ITP which are linked to it, provide the formal substratum into which the appropriate heuristics and/or more formal strategies have to be pre-programmed. This pre-programming is context dependent. The appropriate context is to be generated according to the problem domain or the axiomatic system/theory within which the theorems are to be proved.

It has been demonstrated, that the efficiency in using ITP is gained primarily by entering an appropriate problem dependent heuristics for guiding the ITP [8]. This is the case not only for the 'weighting strategy' inbuilt into the original ITP architecture but also for our FIR scheme. We need to capture the heuristic activity of the agent/problem solver and embed this into the relational structure of the FIR.

Using the Activity Structures based approach to Intelligent Systems [9], we have to describe the problem solving substratum, and transfer it into the virtual substratum structure of the global system composed of cooperating FIR and ITP.

A particular family of strategies can be imposed onto the problem solving substratum by constructing an appropriate protection structure [9], [10] restricting the possible behaviours of the combined ITP/FIR processes. This should be done in a 3-level hierarchy:

- LP 2: The agent-process imposing the restrictions on the possible actions of participants of LP2.
- LP 0: The objects of the problem-solving substratum.

The problem solving substratum (i.e. the architecture) of the system has to be filled with the required problem-knowledge, which is embedded into an appropriate conceptual space, superimposed over the problem-solving substratum. In order to capture both, the logical structure and the heuristics of logical problem solving, a family of appropriate semantic descriptors has to be chosen [10]. The semantic descriptors embedded in a general semantic model based on relational representation of possibilistic structures [11]. These semiotic structures are used to link functionally the theorem proving domain (including appropriate heuristics) with the problem-solving substratum. The normative structure of problem solving determining how the protection restrictions on the activities of the substratum are to be imposed, consists of the following parts:

- The world:
  Abstract world, represented by a model (in mathematical sense) of the problem being solved.

- The conceptual space:
  The logical space in this instance.

- The linguistic world:
  The syntactic means used.

This functional normative structure is equally pertinent to the automatic theorem prover as it is to the activities of a human engaged in the same activity of strategy-directed theorem proving. To capture the intentional nature of a human engaged in this task, we have to couple the normative structure with an appropriate intention structure [9], [10].

References

1. Lusk, E. L. and Overbeek, R.
   *The Automated Reasoning System ITP.*
   Aragonne National Laboratory, Aragonne, IL, 1984.

2. Wos, L., Overbeek, R., Lusk, E., and Boyle, J.
   *Automated Reasoning.*

3. Boyer, R. S. et al.
   Set theory for first order logic: Clauses for Gödel axioms.
4. Bandler, W. and Kohout, L.J.
Special properties, closures and interiors of crisp and fuzzy relations.

5. Kohout, L.J. and Bandler, W.
The use of fuzzy information retrieval techniques in construction of multi-centre knowledge-based systems.

6. Kohout, L.J., Keravnou, E., and Bandler, W.
Automatic documentary information retrieval by means of fuzzy relational products.

Questioning strategy in fuzzy knowledge-based systems.

8. Wos, L.
*Automated Reasoning: 33 Basic Research Problems*.

9. Kohout, L.J.
in press.

10. Kohout, L.J.
Activity Structures: A methodology for design of multi-environment and multi-context knowledge-based systems.
in press.

11. Kohout, L.J.
Theories of possibility: Meta-axiomatics and semantics.
LR/1, A PROLOG-LIKE SYSTEM FOR REASONING WITH IMPRECISE LINGUISTIC INFORMATION*

Hsing-Tai Chung
and
Daniel G. Schwartz
Department of Computer Science and the
Center for Artificial Intelligence
Florida State University
Tallahassee, FL 32306-4019
chung@nu.cs.fsu.edu and
schwartz@nu.cs.fsu.edu

Abstract
This writing outlines an adaptation of SLD resolution, which provides the basis for most implementations of Prolog, to a system developed previously by Schwartz for reasoning with imprecise linguistic information. Since that system employs linguistic concepts not representable in Prolog, it is necessary to extend the syntax and semantics of Prolog to a formalism which is capable of expressing the desired information. This new formalism, together with the appropriate resolution algorithm, will here be referred to as the system for linguistic reasoning LR/1.

Introduction
A central problem in AI research is how to make the machine reason correctly with imprecise information. To date, there have been a number of different proposals as to how this might be accomplished. These include, in particular, the fuzzy logics which have grown out of the work of Zadeh (1975) and the probabilistic and related approaches which have been put forth by Pearl (1988) and Shafer (1976). The present work makes use of still another approach being developed by Schwartz (1989, 1990). This newer reasoning system produces results that are equally intuitively satisfying, and it has the advantage of being algorithmically much simpler.

Key to this new system is an application of the concept of a linguistic variable, first introduced by Zadeh (1975), to the formulation of two new modes of logical inference. These modes of inference differ from those normally associated with “fuzzy logic” in that they do not use fuzzy-set interpretations of the linguistic terms as the basis for defining the meaning of the inference. Instead, the two new types of logical inference deal more directly with the linguistic terms themselves, i.e., as syntactic or symbolic entities. The first of these defines inference along the lines of simple multicriteria decision analysis, using a measure of distance between linguistic terms; the second works with expressions which represent sets of linguistic terms and models logical inference in term of ordinary set inclusion.

Linguistic Variables
A linguistic variable is a variable whose values are expressions in a natural or artificial language (Zadeh 1975). For example, Height may be regarded as a linguistic variable which assumes values from among the linguistic terms {very short, short, medium, tall, very tall}. In (Zadeh 1975) the terms for a linguistic variable are generated by a grammar which allows for composite terms formed by means of various linguistic hedges (e.g., very) and the usual logical connectives, so that every linguistic variable typically has infinitely many linguistic terms. In addition, the terms are provided meanings, given as fuzzy subsets of some underlying domain, e.g., a domain of heights. Thus, formally, a linguistic variable \( \lambda \) is represented as a triple \((T, D, M)\) where \( T \) is a term set; \( D \) is an interpretation domain; and \( M \) is a meaning assignment, given as a mapping which associates a fuzzy subset of \( D \) with each term in \( T \). Reasoning with such terms is then carried out as manipulations of the assigned meanings. Over the years a large number of different interpretations of logical inference, each given as a function defined on fuzzy subsets, have been proposed, see for example (Dubois and Prade 1979, 1988) and (Kandel 1986).
The present treatment builds on a modified version of this idea, developed by Schwartz (1989, 1990), which limits the size of the term set and which seeks to define logical inference in terms of operations that act more directly on the terms themselves, i.e., reasoning is done symbolically rather than semantically. The objective in this has been to avoid the problems of computational complexity associated with the semantics of fuzzy sets. In addition, since reasoning is here done symbolically, it is allowed that the meanings of terms be given also as subintervals of D or possibly as probability distributions over D. It is also allowed that D be empty, in which case it is assumed that terms are assigned subjectively, e.g., saying that X is tall may be simply a matter of opinion, rather than based on an actual measurement of X’s height.

In (Schwartz 1989, 1990) it was assumed that each linguistic variable Λ has an associated primary term λ, e.g., the primary term for Height would be tall. It was then agreed that term sets could have one of six possible forms:

F1: \{ant(λ), λ\}
F2: \{ant(λ), med(λ), λ\}
F3: \{ant(λ), r-ant(λ), med(λ), r-λ, λ\}
F4: \{v-ant(λ), ant(λ), med(λ), v-λ\}
F5: \{e-ant(λ), v-ant(λ), ant(λ), r-ant(λ), med(λ), v-λ\}
F6: \{r-λ, λ, v-λ, e-λ\}

where ant(λ) denotes an antonym of λ, med(λ) denotes an intermediate term of λ, and v, r, and e are abbreviations for the linguistic hedges rather, very and extremely. These are referred to as elementary terms. It is assumed that the elementary terms are ordered in the natural way by a relation ≤. The treatments (Schwartz 1989, 1990) also allow the use of synonyms for elementary terms, e.g., short as a synonym for ant(tall).

Here we shall extend this notion of a linguistic variable to also include term sets which do not have a designated primary term. For example, in what follows, the linguistic variable Position will have the term set

{programmer, systems analyst, project leader, general manager}.

We shall assume that these terms are ordered by ≤ in the manner shown. In what follows, synonyms will be used interchangeably with their corresponding elementary terms.

Two Types of Inference

The first of the two inference modes developed by Schwartz (1989, 1990), here referred to as Mode A, is based on a distance measure δ which is defined by

δ(τ, τ') = g(τ') - g(τ)

where τ and τ' are linguistic terms from the same linguistic variable Λ, and g(τ) and g(τ') are the ranks of τ and τ' respectively. Rankings for terms may be defined in various ways. Here let us assume that the ranks of the terms in term sets of the form F6 are, in the order shown, the integers -4, ..., 4. Then for smaller term sets, we use the corresponding subset of -4, ..., 4. For term sets not having one of the above six forms, ranks will be assigned in a similarly appropriate manner. It follows that, among term sets of the forms F1 through F6, the distance between corresponding terms is always the same, e.g., for any Λ, we have

δ(λ, ant(λ)) = g(ant(λ)) - g(λ) = -2 - 2 = -4

The inference mechanism may now be described as follows. Consider an inference of the form

τ1(X1,1, ..., X1,m1), ..., τn(Xn,1, ..., Xn,mn) ⊃ τ'(X1',1, ..., Xn',n)

where τ1, ..., τn and τ' are linguistic terms from the linguistic variables Λ1, ..., Λn and Λ respectively, and the variables X1',1, ..., Xn',n is a subset of X1,1, ..., X1,m1, ..., Xn,1, ..., Xn,mn. Suppose that for some individuals A1,1, ..., A1,m1 we are given that

τ1(A1,1, ..., A1,m1), ..., τn(A1,1, ..., A1,m1)

are all true, where τ1', ..., τn' are terms from the term sets for Λ1, ..., Λn. Then the inference scheme allows one to conclude τ'(A1',1, ..., A1'm1) where τ' is the term from the term set for Λ which the distance measure δ(τ, τ') is closest to

σ = ∑i δ(τi, τ').

To illustrate, suppose we have two linguistic variables Height and Suitability which have the terms {short, medium, tall} and {unsuitable, fair, suitable} (here writing short for ant(tall), unsuitable for ant(suitable), etc.) Consider the rule

tall(x) ⊃ suitable(x)

and suppose we are given that short(Jim) is true. Then the fact that the distance from tall to short is the same as the distance from suitable to unsuitable yields the conclusion unsuitable(Jim).

The second type of inference developed by Schwartz (1989, 1990), Mode B, uses a set-theoretic interpretation of the linguistic terms. Let Λ be a linguistic variable with an elementary term set E. Then the expressions of Λ are defined inductively as follows: (i) terms in E are expressions of Λ, (ii) if ε and ε' are expressions of Λ, then (ε AND ε'), (ε OR ε') and NOTε are expressions of Λ, and (iii) if τ is a term in Λ, then NOTτ, τ, τ, NOTτ, τ, AT LEAST τ and AT MOST τ are expressions of Λ.

For each expression ε, the relative meaning ρ(ε) is given as a particular subset of E; in particular, ρ(ε) = {ε}, for ε ∈ T, ρ(ε AND ε') = ρ(ε) ∩ ρ(ε'), and ρ(ε OR ε') = ρ(ε) ∪ ρ(ε'). The relative meanings of the remaining expressions are similarly given as subsets of E. The intended meanings of the negations are: NOTτ; τ denotes "anything except τ"; NOTτ, τ denotes "something less than
Consider an inference rule of the form
\[ \varepsilon_1(\alpha_1, 1, \ldots, \alpha_{m_1}), \ldots, \varepsilon_n(\alpha_1, 1, \ldots, \alpha_{m_n}) \Rightarrow \varepsilon(x_1, \ldots, x_n) \]

where \( \varepsilon_1, \ldots, \varepsilon_n \) and \( \varepsilon \) are expressions from the linguistic variables \( \alpha_1, \ldots, \alpha_n \) and \( \Lambda \) respectively, and \( x_1, \ldots, x_n \) is a subset of \( x_1, \ldots, x_{m_1}, \ldots, x_{m_n} \). Let \( \varepsilon_i \) be an expression of \( \alpha_i \), and let \( \alpha_i, 1, \ldots, \alpha_i, n_i \) be individuals. Suppose that \( \varepsilon_i(\alpha_i, 1, \ldots, \alpha_i, n_i) \) is true for \( i = 1, \ldots, n \). For each \( i \), we say that the corresponding hypothesis is satisfied if \( \rho(\varepsilon_i) \subseteq \rho(\varepsilon) \). Under the present inference scheme, if all the indicated hypotheses are satisfied, then one may conclude \( \varepsilon(\alpha_i, 1, \ldots, \alpha_i, n_i) \), where \( \alpha_i, 1, \ldots, \alpha_i, n_i \) are the individuals corresponding to \( x_1, \ldots, x_n \). To illustrate, consider the rule
\[ (AT \ LEAST \ medium)(x) \Rightarrow \text{suitable}(x) \]
and suppose we have that \( \text{tall}(\text{John}) \) is true. Then, since \( \rho(\text{tall}) \subseteq \rho(AT\ LEAST\ medium) \), the given hypothesis is satisfied, and we may conclude \( \text{suitable}(\text{John}) \).

A Prolog-Like System
The present work develops a Prolog-like system where the inferences may be of either of the above two forms, or a combination thereof, where the combined form may be described briefly as follows. Hypotheses in an inference rule are identified syntactically as being associated with an inference of one of the two given types. All hypotheses of type B must be satisfied in order for any conclusion to follow. The conclusion derived (if any) is then based on the type A hypotheses in accordance with inference scheme A.

The syntax, semantics, and resolution method of LR/1 are similar to those of Prolog. We here address the essential differences, specifically, the syntax for atomic formulas and an appropriate adaptation of SLD-resolution.

In LR/1, an atomic formula is a syntactic string of the form
\[ P_\Lambda(x_1, \ldots, x_n) \]
where \( \Lambda \) is the name of a linguistic variable, \( P \) is an expression of \( \Lambda \), thought of as a \( n \)-ary predicate, and \( x_1, \ldots, x_n \) are individual variables. The "\( \Lambda \)" is optional and, if specified, indicates that the formula is to be used for an inference of type A; otherwise it is used as a hypothesis for an inference of type B. For readability and syntactic writeability (on a standard computer), we define
\[ A[\Lambda](x_1, \ldots, x_n ; P) \equiv P_\Lambda(x_1, \ldots, x_n) \]

For example, we shall write \( \text{Height}(\text{John}; \text{tall}) \) for \( \text{tallHeight}(\text{John}) \). In LR/1, inference rules look just like Horn clauses except for the different form of atomic formulas.

It is not hard to show that, if one restricts the term sets of the linguistic variables to have only two terms each (a primary term and an antonym), then the present reasoning system reduces to a simulation of Prolog as it is based on classical logic. It is in this sense that LR/1 may be regarded as an extension of Prolog.

The resolution method for LR/1 is an adaptation of standard SLD-resolution, which provides the logic programming support for the Prolog programming language. This adaptation employs two processes. First is standard unification, which begins by matching on linguistic variables, and then on individual variables and individual terms. Matching ends before the semicolon in the formula (in the rewritten syntax). If the formula is unifiable, then, the expressions after the semicolon are handled by either computation of a distance measure or a check for set inclusion, depending on whether the "\( \Lambda \)" is present or not. The adapted SLD-resolution is complete in the sense that, from an unsatisfiable set of clauses in LR/1, one will always obtain an empty clause (exactly as in standard SLD-resolution) and the final distance measure for a query will be equal to 0.

To illustrate, assume we have a knowledge base \( \Delta \), represented in a Prolog-like form as follows:

1. \( \text{Position}^*(x; \text{project leader}) \leftarrow \text{Suitability}^*(x; \text{suitable}), \text{Leadership}(x; \text{AT LEAST good}). \)
2. \( \text{Suitability}^*(x; \text{suitable}) \leftarrow \text{WorkExperience}^*(x; \text{experienced}), \text{Creativity}^*(x; \text{creative}). \)
3. \( \text{WorkExperience}(\text{Jim}; \text{very experienced}). \)
4. \( \text{Creativity}(\text{Jim}; \text{creative}). \)
5. \( \text{Leadership}(\text{Jim}; \text{very good}). \)

If we have a query like "\( \text{Position}^*(\text{Jim}; x) \)" then we negate it and add it into \( \Delta \) as
\[ \neg \text{Position}^*(\text{Jim}; x). \]

To simplify notations, let \( P, S, L, W, C \) be abbreviations for \( \text{Position}, \text{Suitability}, \text{Leadership}, \text{WorkExperience}, \) and \( \text{Creativity} \). The distance computations for the four terms bearing the "\( \Lambda \)" i.e., \text{Position}, \text{Suitability}, \text{WorkExperience}, and \text{Creativity}, will be denoted respectively by \( d_p, d_s, d_w \) and \( d_c \). The resolution steps are as follows:

7. \( \text{clauses } \) from (7), (8)
   \( \text{unification: unifiable with substitution } \{ \text{Jim}/x \} \)
   \( \text{resolvent: } \neg S(\text{Jim}; \text{suitable}) \lor \neg L(\text{Jim}; \text{AT LEAST good}) \)
   \( d_p = d(\text{Jim}, \text{project leader}) + d_s \)
8. \( \text{clauses } \) from (7), (2)
   \( \text{unification: unifiable with substitution } \{ \text{Jim}/x \} \)
   \( \text{resolvent: } \neg W^*(\text{Jim}; \text{experienced}) \lor \neg C^*(\text{Jim}; \text{creative}) \lor \neg L(\text{Jim}; \text{AT LEAST good}) \)
   \( d_w = d(\text{Jim}, \text{suitable}, \text{suitable}) + d_w + d_c \)
(9) clauses from : (8), (3)
unification : unifiable
resolvent : $\neg C'(\text{Jim};\text{creative}) \lor$
$\neg L(\text{Jim}; \text{AT LEAST good})$
distance : $d_s = d(\text{experienced, very experienced}) = 1$

(10) clauses from : (9), (4)
unification : unifiable
resolvent : $\neg L(\text{Jim}; \text{AT LEAST good})$
distance : $d_s = d(\text{creative, creative}) = 0$

(11) clauses from : (10), (8)
unification : unifiable
resolvent : empty clause
set inclusion : true

In (11), an empty clause has been derived, thus completing the unification process. Next we have to compute all the distance measures in reverse order. First we get $d_s = 1$. Then we set $d_p = 0$ (in order to complete the resolution), from which $d(\varepsilon, \text{project leader}) = -1$. The resulting term $\varepsilon$ then will be the general manager according to the actual term set for the linguistic variable Position shown previously.

Concluding Remarks

The system LR/1 is an extension of Prolog that allows for expressing, and reasoning with, certain types of imprecise or incomplete information. Future work will extend the capabilities of LR/1 by allowing for more complex structures and operations on the elementary terms for a linguistic variable and for defining relations between linguistic variables in order to build a higher level knowledge representation scheme. Another separate but related, project will be to investigate the possibility of extending first-order logic in a manner analogous to that in which LR/1 is an extension of Prolog. This will involve in particular developing a corresponding generalization of one or more of the known resolution methods for full first-order systems.

References


PERFORMANCE-DRIVEN KNOWLEDGE TRANSFORMATION

Lawrence B. Holder
Department of Computer Science
University of Illinois
Urbana, IL 61801

ABSTRACT
Integration of machine learning methods with knowledge-based systems requires sophisticated control mechanisms for applying methods appropriate to the performance task. Performance-driven knowledge transformation controls the application of learning methods based on their ability to achieve desired performance goals while preserving the performance on other tasks. A means-ends approach to performance-driven knowledge transformation is presented along with experimental results from an implementation. The results indicate that performance-driven knowledge transformation is able to maintain multiple performance goals by applying appropriate machine learning methods to transform a knowledge base.

INTRODUCTION
Intelligent systems acquire knowledge in order to perform some task. The level of performance depends upon the quality of the acquired knowledge. Machine learning methods attempt to improve the quality of the knowledge, and thus improve performance on the desired task. Integration of machine learning methods with knowledge-based systems will reduce the dependency of system performance on the quality of knowledge initially entered by the knowledge engineer. For example, instead of extracting high-level rules from a domain expert, the knowledge engineer might need to collect only previous instances with known conclusions. The machine learning component of the knowledge-based system may then generalize the instances into knowledge of the appropriate quality.

Several machine learning methods have been developed that successfully derive higher-level knowledge from examples. However, these methods work in isolation and do not consider the need for multiple learning methods within a single knowledge-based system. For instance, if the knowledge takes the form of examples and the goal is improved accuracy, then the system may choose to invoke an empirical learning method that constructs generalized rules describing the examples. On the other hand, if the knowledge takes the form of higher-level rules and the goal is improved response time, then the system may choose an analytical learning method to construct a macro from the rule inference chain used to solve the current problem. Knowledge-based systems must be able to select learning methods appropriate for the desired performance improvement. In addition, application of the learning method must preserve the performance goals of other knowledge used for other tasks.

Knowledge-based systems need the capability of applying multiple learning methods in order to adapt to new performance requirements while maintaining previously stated goals. For example, once a certain level of accuracy has been achieved, the emphasis may shift to the time needed to arrive at an accurate decision. The desired accuracy may have been achieved by rote learning examples, and then the desired speed may be achieved by generalizing these examples to a few general rules. Current individual learning systems are incapable of a corresponding shift of attention.

Performance-driven knowledge transformation controls the application of learning methods based on their ability to achieve desired performance goals while preserving the performance on other tasks. The next section describes work related to performance-driven knowledge transformation. Then, the approach is discussed in detail. Next, results are presented from preliminary experimentation with an implementation of the approach in the PEAK system.

RELATED WORK
Experimentation with empirical learning programs has shown that more emphasis should be placed on performance during the construction of the learned concepts. Evidence for this can be found in the results obtained with the decision trees generated by Quinlan's ID3 program (Quinlan 1986). Quinlan found that pruning the rules extracted from a decision tree can improve the accuracy of the rules on unseen examples (Quinlan 1987). Compared to the original rules, the pruned rules performed better on
a set of unseen test examples. Recent experiments by Mingers have confirmed this result (Mingers 1989).

Further evidence for the necessity of performance for guiding concept learning can be found in experimentation on explanation-based learning systems. In experimentation with the PRODIGY system, Minton found that performance degrades as the number of rules grows large (the utility problem) (Minton 1988). In order to learn a concept, the system acquires several rules whose disjunction forms the system's understanding of the concept. As the number of rules increase, the cost of determining the applicability of a rule may outweigh the benefits of applying, and thus, retaining the rule. Minton's solution is to maintain empirical estimates of match costs, application savings and frequency of application for each rule. These estimates are used to compute a utility value for the rule. If this value becomes negative, the rule is no longer considered. Minton found that maintenance of a rule's utility value and compression of the rule's conditions result in a substantial performance improvement.

This related research illustrates that learning should be constrained by the desired performance. If the knowledge base satisfies the performance goals, then no learning is necessary. If performance is violated, then only enough learning should be done to achieve the performance without violating previously satisfied performance goals.

**APPROACH**

Performance-driven knowledge transformation controls the application of learning methods based on their ability to achieve desired performance goals. Each task for the knowledge base defines a performance space. The dimensions of the performance space are the performance goals (e.g., completeness, correctness, response time) to be maintained by the knowledge base for that task. The current state of the knowledge base is represented by a point in the performance space for each task. A knowledge transformation can be viewed as a move of the current knowledge base from one point in the performance space to another. Figure 1 shows the performance space for one task. The task consists of three performance goals $G_1, G_2$ and $G_3$. The location of two knowledge bases $K_1$ and $K_2$ are shown for the task.

The desired performance for each task defines a hyper-rectangle in that task's performance space. When the knowledge base moves outside the performance hyper-rectangle in some performance space, performance-driven knowledge transformation selects a learning method to transform the knowledge base so that the corresponding point in the performance space for the current task moves inside the desired performance hyper-rectangle without moving the points for other tasks outside the desired hyper-rectangle in the performance spaces. Referring to Figure 1, knowledge base $K_1$ violates the performance goals for the task. Transforming knowledge base $K_1$ into $K_2$ achieves the desired performance goals.

This research investigates a means-ends approach to performance-driven knowledge transformation. When a performance goal violation is detected while solving a problem from some task, the means-ends approach uses information about the context of the goal violation (e.g., the difference between desired and actual performance) to select a transformation operator for reducing this difference while maintaining other performance levels. Application of the operator yields a new knowledge base. If the new knowledge base achieves the violated performance goals and preserves other performance goals, then the current knowledge base is replaced by the new knowledge base. Otherwise, another transformation operator is selected for application.

In the following discussion, certain assumptions have been made about the knowledge in the knowledge base and the performance element using this knowledge. The knowledge base is a set of Horn clause rules. The performance element is a deductive retriever similar to Prolog. Performance is measured while the performance element attempts to solve a query posed by the user. Attached to the query are the performance goals to be maintained during solution. Performance goal violations occur when the measured performance exceeds the desired thresholds.

**Performance Perspective**

Using performance goals as a means of guiding the maintenance and repair of a knowledge base requires a precise definition of performance. The definition of performance depends on the perspective. Four perspectives are applicable for describing the performance of a knowledge base:
- **External** performance is the performance measured from outside the knowledge base, regardless of any internal knowledge transformations.

- **Current** performance is the performance the system currently maintains for the previously seen queries.

- **Expected** performance is the performance the system expects to demonstrate on future queries. Expected performance is usually the same as current performance.

- **Absolute** performance is the performance that the current state of the knowledge would support if given every possible query.

When the user specifies a threshold for some performance measure, the proper perspective must be used to evaluate the performance of the knowledge base. **Absolute** performance is rarely available due to a lack of knowledge about the instance space. **Absolute** performance is inappropriate, because the distribution over the entire instance space may not give equal probability to each instance. **External** performance provides information about the rate of convergence towards absolute performance. Changes in **external** performance indicate the need for an increase or decrease in the extent of the knowledge transformations. **Current** performance evaluates the knowledge only on previously seen queries. **Expected** performance is the best measure of the current state of the knowledge base, because the objective of the knowledge base is to maintain its expected ability to perform the task within desired thresholds on possibly unseen queries.

Performance-driven knowledge transformation should measure both expected and external performance. Knowledge transformations are triggered only when **expected** performance falls below desired levels. **External** performance should then be used in the selection of an appropriate transformation operator. The greater the difference between **external** and expected performance, the more drastic a transformation operator should be recommended by the system.

**Information on Goal Violations**

Once a goal violation has been detected, several pieces of information are available for selecting an appropriate knowledge transformation operator. First, as described in the previous section, the difference between expected and external performance indicates the extent of the necessary transformation.

Second, after the performance element attempts to solve a query, the violated and preserved goals are known. Each goal contains information about the performance measure that this goal constrains, the desired threshold on the measure, the observed value of the measure on previously seen queries (including the query just processed), and the difference between the observed and desired performance (the error). The performance measure constrained by a violated goal is useful for selecting transformation operators capable of improving this performance measure. The magnitude of the error indicates the extent of the transformation. The performance measure constrained by a satisfied goal is useful for selecting transformation operators capable of preserving this performance measure. The magnitude of the error indicates the extent to which the selected operator may degrade performance on the satisfied goals in order to achieve performance on the violated goals.

A third source of information that will be available upon detection of a performance goal violation is the **task history**. Each task known to the knowledge base maintains a task history of previously seen queries from the task. The task history serves two purposes. First, the task history represents an empirical estimate of the distribution over the possible queries of the task. This distribution can be used to verify the achievement of violated performance goals in transformed knowledge. Second, an entry in the task history contains information about the query-solving episode. One useful piece of information about a query-solving episode is the trace of the knowledge accessed during the solution.

The **knowledge trace** is an and/or tree that records the knowledge accessed during the solution of the query and indicates which rules (if any) support the response to the query. Information about the shape of a task's knowledge traces constrains the selection of knowledge transformations. For example, wide, shallow knowledge traces indicate that the knowledge consists of specific instances of the task; whereas narrow, deep knowledge traces indicate a more general set of rules for proving queries from the corresponding task.

Finally, past success of the transformation operators provides information upon performance goal violation. As the knowledge base transforms to meet performance goals, a record is kept of the old and new knowledge bases along with the operator responsible for the transformation. If the new knowledge base achieves a violated goal while preserving non-violated goals, then the system increases the operators applicability for achieving and preserving the appropriate goals. Over time, collection of this information will allow the system to make a more informed operator selection based on past experience.

**Verification of Knowledge Base**

Because no operator application is guaranteed to achieve the desired results, the system must verify that the knowledge base resulting from an operator application
achieves the desired performance. Verification can be accomplished by re-solving the queries in the task history. The size of the task history can be changed to tradeoff performance convergence rates for transformation speed. As the system learns operator applicability, there is less chance of multiple verification being necessary to repair one goal violation; thus, the task history size can be increased over time.

EXPERIMENTATION

This section illustrates an experiment conducted with the PEAK system. The experiment follows the shuttle landing control database available from the machine learning databases maintained by the University of California at Irvine. The problem is to determine whether to land the shuttle manually or automatically based on environmental attributes. The corresponding task is labeled the landing task, and the queries are of the form landing(ENV, ?x). The ENV in the query represents the environmental situation to be evaluated. The performance element attempts to fill in the ?x with the recommended landing control: auto or noauto.

Prior to query answering, the user inputs the performance thresholds to be maintained by the knowledge base while answering landing queries using the performance element (a backward-chaining deductive theorem prover for Horn clauses). For this experiment, three performance goals are specified: correctness, completeness and response time. The correctness goal specifies that the answers to queries must be correct 90% of the time. The completeness goal specifies that the query must be answered 95% of the time. That is, the answer should be either auto or noauto and not "I don't know". The response time goal specifies that the performance element must respond within 10 seconds.

Two knowledge transformation operators are available: rote learning and empirical learning. Application of the rote learning operator asks the user for the correct answer to the query. A new rule is added to the knowledge base having the instantiated query as the consequent, and the facts defined before query execution as the antecedent. The empirical learning operator utilizes the ID3 program to build a decision tree from examples in the knowledge base. The examples are rules such as those learned by the rote operator. Each path in the resulting decision tree is converted to a rule. The examples are replaced by the new rules in the transformed knowledge base.

Starting with an empty knowledge base, PEAK attempts to solve landing queries, while maintaining the performance goals. Figure 2 plots the three performance goals for 200 randomly chosen queries from the shuttle landing control domain.

Figure 2a illustrates how PEAK maintains response time performance below 10 seconds. For the first 30 queries, response time increases as the number of rote-learned rules increases. Eventually, the large number of rules in the knowledge base cannot be traversed within the response time threshold.

While processing the 30th query, PEAK was unable to solve the query, generating a completeness failure. PEAK first tries to transform the knowledge base by rote-learning...
a new rule. However, verification of the new knowledge base uncovered a response time failure. Because the route learning operator was ineffective, PEAK chose to apply the ID3 operator. ID3 generalized the 29 learned instances into 8 general rules. As Figure 2a indicates, the resulting transformation drastically improves response time performance.

The plot of completeness performance in Figure 2b illustrates how PEAK quickly learns the initial query knowledge. After the ID3 transformation, completeness remained above the 95% threshold for the remainder of the 200 queries.

The correctness plot in Figure 2c shows how performance starts at 100% and converges to the desired 90% threshold. The initial values of 100% for correctness are due to the fact that many of the initial queries could not be answered. Correctness performance only measures the correctness of answered queries. Immediately following the application of ID3, correctness falls to 94% due to the next two queries being incorrectly answered according to the new knowledge base. As query answering continues, the over-generalization in the rules eventually brings correctness down below the 90% threshold. Correctness violations occur at queries 99, 98, 153 and 163. In each case, PEAK uses the route-learning operator to memorize the incorrectly answered query and restore 90% correctness performance.

The final knowledge base after completion of the 200 queries consists of the 12 rules shown below.

1. landing(z,noauto) ← sign(z,nn) & wind(z,head) & stability(z,xstab) & error(z,MM) & magnitude(z,Medium) & visibility(z,yes)
2. landing(z,noauto) ← sign(z,pp) & wind(z,tail) & stability(z,xstab) & error(z,MM) & magnitude(z,Low) & visibility(z,yes)
3. landing(z,noauto) ← sign(z,nn) & wind(z,head) & stability(z,xstab) & error(z,MM) & magnitude(z,OutOfRange) & visibility(z,yes)
4. landing(z,noauto) ← sign(z,nn) & wind(z,tail) & stability(z,xstab) & error(z,MM) & magnitude(z,Low) & visibility(z,yes)
5. landing(z,auto) ← error(z,MM) & visibility(z,yes)
6. landing(z,auto) ← stability(z,xstab) & error(z,SS) & magnitude(z,Strong) & visibility(z,yes)
7. landing(z,auto) ← visibility(z,no)
8. landing(z,noauto) ← error(z,XL) & visibility(z,yes)
9. landing(z,noauto) ← error(z,LX) & visibility(z,yes)
10. landing(z,noauto) ← stability(z,xstab) & error(z,SS) & magnitude(z,Strong) & visibility(z,yes)
11. landing(z,noauto) ← error(z,SS) & magnitude(z,OutOfRange) & visibility(z,yes)
12. landing(z,noauto) ← error(z,SS) & magnitude(z,Low) & visibility(z,yes)

Rules 5-12 are the general rules learned by ID3. Rules 1-4 are the specific instances learned to repair the over-generalization in ID3's rules. After 200 queries, the knowledge base converged to 8 general rules describing major trends in the shuttle landing domain and four specific rules for special cases not handled correctly by the general rules.

One final observation from Figure 2 is the convergence of the performance towards the desired thresholds and not towards the maximum possible performance. This indicates how performance-driven knowledge transformation utilizes flexibility in one dimension of performance to improve performance in another dimension.

CONCLUSION

The goal of knowledge acquisition is the ability to perform some task. The goal of learning is to improve performance on the task. Performance-driven knowledge transformation controls the application of learning methods to maintain the desired performance levels of the knowledge base. Future experimentation with the PEAK system will analyze the convergence properties of a knowledge base with respect to the available transformation operators and the performance goals. The results will provide both theoretical and applied mechanisms for the integration of learning methods with knowledge-based systems.

ACKNOWLEDGEMENTS

I would like to thank Robert Stepp, Diane Cook, Brad Whitehall and Robert Reineke for their helpful suggestions on this work.

REFERENCES


A FORMAL CONSTRUCTIVIST MODEL OF KNOWLEDGE REVISION

Ronald R. Yager
Machine Intelligence Institute
Iona College
715 North Avenue
New Rochelle, NY 10801

Kenneth M. Ford
Institute for Human & Machine Cognition
Division of Computer Sciences
The University of West Florida
11000 University Parkway
Pensacola, Fl. 32514

ABSTRACT

We will discuss a formal model of human and machine learning called participatory learning (Yager, 1990). This model enables us to represent machine learning and belief revision in a constructivist framework. In this model, the learner's previous beliefs play an important role in the assimilation of further information. A central aspect of the theory is the degree of compatibility between observations and belief. In addition, the role of arousal or anxiety (which occurs when we are continuously confronted with data that conflicts with our beliefs) is discussed.

INTRODUCTION

For most of this century associationist or behaviorist theories of learning dominated psychology in the United States. However, the rigid and prescriptive nature of associationist psychology precluded an adequate general description of human learning, and specifically, the means by which scholars produce knowledge (Novak, 1987). Unfortunately, most efforts at machine learning and cybernetic epistemology remain rooted in the behaviorist tradition.

Personal construct theory, as formulated by Kelly (1955, 1969, 1970) and summarized by Adams-Webber (1987), assumes that people typically use cognitive dimensions termed 'constructs' to evaluate their experience. Kelly's model of the personal scientist implies that each of us seeks to predict and control events by forming relevant hypotheses, and then testing them against available evidence. In other words, we humans frequently anticipate the occurrence or non-occurrence of future events based on our willingness to project observed uniformities into the future. Thus, we continually glide from the past into the future with our previous experience preceding us — illuminating and organizing the manner in which subsequent events will be manifest to us (Ford, 1989).

In a constructivist theory, learning is a bootstrap process in the sense that you learn and revise your beliefs within the framework of what you already know or believe. In fact, in the epigraph to his influential text, Educational Psychology: A Cognitive View, Ausubel et al. (1978) stated:

"If I had to reduce all of educational psychology to just one principle, I would say this: The most important single factor influencing learning is what the learner already knows. Ascertain this and teach him accordingly."

The name participatory learning highlights the fact that the learner's current knowledge of the subject participates intimately in the learning process. A prototypical example of participatory learning is that of trying to convince a scientist to discard an old theory for a new one. In this situation we must relate and explain the new theory in terms of the scientist's view of the old theory. Thus the old theory (as construed by the scientist) participates in the learning of the new theory. Central to participatory learning is the idea that an exogenous observation has the greatest impact on learning (i.e., revision of belief) when the observation is largely compatible with our present belief system. In particular, observations in conflict with our current core constructs (i.e., strongly held beliefs) are discounted. These core constructs serve as hidden hand editors; they are robust in the face of all but the strongest anomalous or discrepant feedback. Such powerful implicit feedforward mechanisms not only help to predetermine the knowledge we construct or "discover", but also aid in maintaining and defending it (Agnew, 1989).
The formal model discussed below will reflect the notion that participatory learning is optimal in situations in which it is necessary to change only a small part of the learner’s current belief system. Informally, we can say that an intelligent reasoner endowed with the capacity for participatory learning employs sympathetic observations to modify itself. Occasional, very unsympathetic observations are discounted as erroneous.

A PARTICIPATORY LEARNING PROCEDURE

Assume \( k = 1, \ldots, q \) indicates a collection of nodes or variables. Let \( V_k \) be a valuation associated with each node. It is our objective to learn these valuations. We shall assume that \( V_k \in [0, 1] \). We shall also assume that our knowledge about these values comes in a sequence of observational vectors, \( D(1), D(2), \ldots, \) each of dimension \( q \). Thus \( D(j) \in [0, 1] \) is the manifestation of the \( k \)-th valuation in the \( j \)-th observation. We are using the \( D \) vectors as a means of learning about the valuations, the \( V \)'s. We shall say that our learning process is participatory if the usefulness of each of the observations, \( D(j) \), in contributing to the learning process depends upon its acceptance by the current estimate of the \( V \)'s as being a good observation. Implicit in this situation is the idea that for an observation \( D(j) \) to be useful in learning about the \( V \)'s it must in some sense agree with the current estimate of the \( V \)'s. \( V_k(j) \) is our estimate for \( V_k \) after \( j \) observations. We shall let \( V(j) \) indicate the vector of these \( q \) values. Then our idea of participatory learning means that for \( D(j) \) to be useful (or used) in helping estimate the \( V \)'s, \( D(j) \) should in some sense agree with \( V(j) \). The formal mechanism we will use for updating our estimate (or belief) is a smoothing-like algorithm (Brown, 1963), i.e.,

\[
V(j + 1) = V(j) + \alpha \cdot p_j \cdot (D(j) - V(j)).
\]

From the above: \( V(j + 1), V(j), \) and \( D(j) \) are \( q \)-vectors corresponding respectively to the new belief, the old belief, and the current observation. Furthermore, \( \alpha \in [0, 1] \) is the base or primary learning rate and \( p_j \) which is the current compatibility level, is also required to satisfy \( p_j \in [0, 1] \). In a participatory learning situation \( p_j \) is dependent upon the compatibility between the current belief and the current observation, thus we require \( p_j = F(D(j), V(j)) \) where \( p_j = 0 \) indicates complete incompatibility and \( p_j = 1 \) complete compatibility. In the above, if the current observation, \( D(j) \), is completely contradictory with our current belief system \( V(j) \) then \( p_j = 0 \) and \( V(j + 1) = V(j) \).

Thus, in the case of complete contradiction the system discounts the current observation (i.e., it is not open to any learning from the current observation). Effectively, the system assumes \( D(j) \) is not a valid observation. Reflecting the other extreme is the situation in which \( p_j = 1 \). In this case the observation is in complete agreement with the current belief system. Accordingly, with \( \alpha = 1 \) we get

\[
V(j + 1) = V(j) + 1 \cdot (D(j) - V(j)), \quad V(j + 1) = D(j),
\]

and thus our system is the most open to the new information.

Since \( p_j \) is a measure of the agreement or compatibility of the current observation \( D(j) \), with the current belief \( V(j) \), a possible formulation for \( p_j \) is

\[
p_j = 1 - 1/4 \sum_{k=1}^{q} |D_k(j) - V_k(j)|
\]

Thus in the above, \( p_j \) is the complement of the average absolute difference between each observation and its corresponding current belief. (Alternatively we could use the sum of the differences.)

Using the above formulation for \( p_j \) we get for each individual node, \( m = 1, \ldots, q \), \( V_m(j + 1) = V_m(j) + \alpha \cdot (D_m(j) - V_m(j)) \cdot (1 - (1/4 \sum_{k=1}^{q} |D_k(j) - V_k(j)|)) \)

Example: Assume \( q = 4 \) and \( \alpha = 1 \). Let the situation be as described below:

\[
V(j) = \begin{bmatrix} 0.7 \\ 0.3 \\ 0.2 \\ 0.3 \end{bmatrix}, \quad D(j) = \begin{bmatrix} 0.7 \\ 0.5 \\ 0.5 \\ 0.3 \end{bmatrix}
\]

Hence \( D(j) - V(j) = \begin{bmatrix} -0.3 \\ 0.2 \\ 0.3 \\ 0.1 \end{bmatrix} \) and

\[
p_j = 1 - 1/4 \sum_{k=1}^{4} |D_k(j) - V_k(j)| = 0.7.
\]

Thus the new updated belief \( V(j + 1) \) is

\[
V(j + 1) = V(j) + 0.7 \cdot (D(j) - V(j)),
\]

\[
V(j + 1) = \begin{bmatrix} 0.7 \\ 0.3 \\ 0.2 \\ 0.3 \end{bmatrix} + 0.7 \begin{bmatrix} -0.3 \\ 0.2 \\ 0.3 \\ 0.1 \end{bmatrix} = \begin{bmatrix} 0.35 \\ 0.51 \\ 0.57 \\ 0.31 \end{bmatrix}.
\]

Parenthetically it should be noted that while the case when \( p_j = 1 \) induces an environment where the system is most conducive to learning—no learning occurs in this case. For we note that for \( p_j \) to equal 1 it must be the case:

\[
1/4 \sum_{k=1}^{q} |D_k(j) - V_k(j)| = 0 \quad \text{and hence for all } k = 1, \ldots, q \quad D_k(j) - V_k(j) = 0.
\]
Thus \( V_m(j + 1) = V_m(j) + \alpha \cdot 1 \cdot (D_m(j) - V_m(j)) \), and \( V_m(j + 1) = V_m(j) \). As we shall subsequently see the most dramatic belief revision occurs when the vectors \( D(j) \) and \( V(j) \) are "close".

Considering the basic learning algorithm for a particular node \( k \):

\[
V_k(j + 1) = V_k(j) + p_j(D_k(j) - V_k(j)),
\]

the most dramatic learning for this node occurs when

\[
\Delta_k(j) = p_j \cdot 1 \cdot D_k(j) - V_k(j) \cdot 1 \cdot \alpha
\]

assumes its maximal value. Without loss of generality we shall assume \( \alpha = 1 \).

Since \( p_j = 1 \cdot 1/q \cdot \sum_{m=1}^{q} |D_m(j) - V_m(j)| \)

and denoting \( |D_m(j) - V_m(j)| \cdot 1 = e_m(j) \),

we get

\[
\Delta_k(j) = e_k(j) \cdot 1 \cdot (1 - 1/q \cdot \sum_{m=1}^{q} e_m(j)).
\]

Rewriting this as

\[
\Delta_k(j) = e_k(j) \cdot 1 \cdot (1 - 1/q \cdot e_k(j) + \sum_{m \neq k} e_m(j))
\]

and denoting \( e_k(j) \) as \( a \) and \( \sum_{m \neq k} e_m(j) \) as \( b \),

we can express the above as

\[
\Delta_k = a \cdot (1 - (1/a \cdot 1/qb)) = (a - 2/a \cdot b)q
\]

where \( a \in [0, a_1] \) and \( b \in [0, b_2] \)

where \( a_1 = \text{Max}(1 - V_k(j), V_k(j)) \) and

\[
b_2 = \sum_{k \neq m} \text{Max}(1 - V_m(j), V_m(j)).
\]

It can be easily shown that the maximum of \( \Delta_k \) occurs at

\[
\sum_{k \neq m} e_m(j) = 0, \quad e_m(j) = a_1 = \text{Max}(1 - V_k(j), V_k(j)).
\]

At this point \( \Delta_k = a_1 \cdot (1 - a_1 / q) \). Thus the maximal learning for a particular node occurs when all the other nodes are in agreement and the node in question is the most distinctive from its current value. The essential idea we see is that maximal learning occurs in an incremental step-wise fashion. In the above we see that the primary rule \( \alpha \) is essentially "modulated" by the compatibility. It is possible to envision a more general formulation of the participatory learning algorithm, e.g.,

\[
V(j + 1) = V(j) + \beta_j \cdot (D(j) - V(j))
\]

where \( \beta_j \) is some, perhaps nonlinear, function of \( p_j \) such that (1) \( \beta_j \in [0, 1] \) and (2) \( \partial \beta_j / \partial p_j \geq 0 \). Condition 2 implies that as \( p_j \) increases the learning rate \( \beta_j \) increases.

**FORMULATION OF THE COMPATIBILITY**

In the previous section we introduced a measure of compatibility, \( p_j \), between the current belief system \( V(j) \), and the current observation. We will now discuss more general alternative formulations of this measure. In order to simplify the notation we shall drop the sequence number parameter \( j \). Thus \( D_k \) and \( V_k \) shall indicate the current belief and current observation of variable \( k \), while \( D \) and \( V \) correspond to the associated vectors.

The term \( 1 - |D_k - V_k| \) can be seen to be a measure of similarity between \( D_k \) and \( V_k \). We will use the term \( S_k \) to indicate the measure of similarity between \( D_k \) and \( V_k \), where \( S_k \in [0, 1] \). Formally, we can introduce a function \( G_k \) such that \( S_k = G_k(D_k, V_k) \). That is, \( G_k \) maps \( (D_k, V_k) \) into a degree of similarity. Introduction of \( G_k \) allows us to free the \( D_k \)'s and \( V_k \)'s from being in the unit interval. Secondly, it provides the facility for the two \( D_k \) and \( V_k \) to have an \( S_k = 1 \) even if they are not exactly equal. Thirdly, it allows for different perceptions of similarity for different nodes (different \( k \)'s). We shall let \( S \) be the vector of similarities. Thus, \( p = F(S) = F(S_1, S_2, \ldots, S_q) \). Essentially, \( F \) is seen to be an aggregation function. A fundamental property that must be associated with \( F \) is that it is monotonic. This requires that if \( a_j \geq b_j \) for all \( j = 1, \ldots, q \) then \( F(a_1, a_2, \ldots, a_q) \geq F(b_1, b_2, \ldots, b_q) \). A second property is that \( F(1, 1, \ldots, 1) = 1 \). This condition implies that if each element is similar there is overall similarity.

More generally we recall that in a participatory learning (or revision) algorithm the parameter \( p \) measures the degree to which our observed data \( D \) agrees with our current belief. In formulating such a function we have some latitude in defining how restrictive we are in conceptualizing what we mean by the idea of agreement. That is, do we require that all the \( D_k \)'s be in agreement with the \( V_k \)'s (i.e., all the \( S_k \)'s are 1) or are we willing to accept a situation in which most of the nodes are similar. In order to provide the most latitude in representing this concept we introduce a class of aggregation operators called OWA (ordered weighted averaging) operators to formulate the \( F \) function. This operator was originally introduced by Yager (1988) in the framework of multiple criteria decision functions.

**Definition:** A mapping \( F \) from \( R \to R \) where \( I = (0, 1) \) is called an OWA operator of dimension \( q \) if associated with it is a weighting vector \( W \): \( W = (W_1, W_2, \ldots, W_q) \) such that: (1) \( W_j \in [0, 1] \) and (2) \( \Sigma W_j = 1 \) where \( F(S_1, S_2, \ldots, S_q) = W_1 B_1 + W_2 B_2 + \ldots + W_q B_q \) where \( B_j \) is the \( j \)th largest element in the collection.
It is important to emphasize that the weights in an OWA operator are associated with a particular ordered position rather than a particular $S_k$.

Example: Assume $F$ is an OWA operator of size 4 with $W = [0.2, 0.3, 0.1, 0.4]$.

If $S_1 = 0.6$, $S_2 = 1$, $S_3 = 0.3$, and $S_4 = 0.5$ then the $b$'s are

$$b_1 = 1, b_2 = 0.6, b_3 = 0.5, b_4 = 0.3;$$ hence,

$$p = (0.2)(1) + (0.3)(0.6) + (0.1)(0.5) + (0.4)(0.3) = 0.55.$$

It is important to note that our original formulation for $p$ is a special OWA operator in which $W_j = 1/q$ for all $j$. The elements of $W$ can be associated with an indication of how open we are to learning. More formally, we can associate with a given OWA operator, $W$, a measure $q$, i.e.,

$$\text{PAR}(W) = 1 - ((1/q - 1) \cdot \sum_{k=1}^{q} ((q - 1) \cdot W_k))$$

indicating the degree of participation required. In particular, with $\text{PAR} \in [0, 1]$, $\text{PAR}(W) = 1$ indicates a requirement for complete agreement of all the individual nodes before learning. $\text{PAR}(W) = 0$ indicates a willingness to learn with minimal participation agreement. Appropriate selection of the weights allow us to emulate environments in which we can specify the degrees of agreement required from the constituent nodes.

**Anxiety as Background Arousal**

As indicated earlier, in this participatory learning model a high compatibility (manifested by $p_j$) between the current belief and the current observation enhances the environment for learning. We now introduce the facility for an opposing phenomenon. From a Kelyan vista, salient and/or massed negative feedback is the source of individual anxiety. In particular, in the type of learning environment we are trying to emulate, if over a long period we see a procession of low $p_j$'s (i.e., incompatibility of the belief and observations) the system should become more open to learning. The background process we are trying to capture is essentially saying that if we see a long string at low compatibilities between our belief and the data, we may come to believe that our belief structure is wrong (not the data). This is seen as a type of arousal. We shall let $a_{j+1} \in [0, 1]$ be our arousal index. The higher the $a_j$, the more aroused we are. In particular:

$$a_{j+1} = a_j + \beta \cdot (1 - p_{j+1}) - a_j.$$

Thus if $p_{j+1} = 1$, high compatibility $a_{j+1} = a_j - \beta a_j$; the arousal rate decreases. On the other hand $p_{j+1} = 0$ requires $a_{j+1} = a_j + \beta (1 - a_j)$ and thus causes an increase in $a$. The value of $\beta \in [0, 1]$ indicates the rate of arousal, i.e., the closer $\beta$ is to one — the quicker the system is to sense the background process. A model of the participatory learning process that enables us to appropriately include the background arousal process is

$$V(j + 1) = V(j) + (\alpha \cdot p_j)^{1-a_j} \cdot (D(j) - V(j)).$$

Here we see that as our arousal increases, the term $(\alpha \cdot p_j)^{1-a_j}$ increases.

More generally we can indicate the learning model as

$$V(j + 1) = V(j) + G(a_j, p_j) \cdot (D(j) - V(j))$$

where in the above $G(a_j, p_j) = (\alpha \cdot p_j)^{1-a_j}$.

An alternate form for $G$ is $G(a_j, p_j) = \lambda \cdot (\alpha \cdot p_j)^{1-a_j}$.

In this formulation, $\lambda \cdot \alpha$ is our learning rate in a completely compatible environment ($a_j = 0$ and $p_j = 1$) while $\lambda$ rate is our learning when completely aroused. An alternative formulation for $G$ is

$$G(a_j, p_j) = (a_j + \alpha \cdot p_j - a_j \cdot \alpha \cdot p_j).$$

Note that when arousal is complete

$$a_j = 1 \text{ and } G(a_j, p_j) = \lambda,$$

but when arousal is minimum

$$a_j = 0, p_j = 1, \text{ and } G(a_j, p_j) = \lambda \cdot \alpha.$$

**Summary**

The main focus of this paper has been a discussion of recent research directed toward both the development of a formal model of participatory learning and the elaboration of its constructivist theoretical rationale.

It was noted at the outset that most artificial intelligence research efforts (especially those aimed at addressing issues in machine learning) harbor implicit behaviorist assumptions. In contrast, as we have suggested elsewhere (Ford and Adams-Webber, 1990), an explicit constructivist model of human cognitive processes, including perception, representation, anticipation, and reasoning, might provide useful conceptual supports (guides) for work in machine learning and belief revision.

We introduced a formal model of human and machine learning called participatory learning, in
which the learner's previous beliefs play an important role in the assimilation of further information. Its theoretical rationale was explicated in terms of some of the basic tenets of personal construct theory and assimilation theory.

In addition, the role of arousal or anxiety (which occurs when we are continuously confronted with data that conflicts with our beliefs) is discussed.

ACKNOWLEDGEMENTS

We owe particular thanks to Jeff Yerkes for the appearance of this paper.

REFERENCES


EXPERIMENTS IN ROUTING AN AUTONOMOUS LAND VEHICLE WITH A WEAKLY INDUCTIVE LEARNING ALGORITHM

Mikel D. Petty, Terry J. Frederick, and J. Michael Moshe1
Department of Computer Science
University of Central Florida
Orlando FL 32816

ABSTRACT

Pattern matching and induction are well-known techniques in machine learning. This paper presents a machine learning algorithm based on these techniques that has been applied to the problem of routing an autonomous land vehicle over terrain. A set of experiments with the algorithm are analyzed.

INTRODUCTION

Routing an autonomous land vehicle (ALV) over terrain is a problem of current interest. In many routing systems, a fixed algorithm selects a route after being given the terrain and routing goals as input (e.g. see [Stanzione, 1989]).

We revisit a simple approach to machine learning, and test its applicability to the ALV routing problem. The learning algorithm is used to select the route of an ALV through unfamiliar and dangerous terrain. (Both the ALV and the terrain are simulated.) Starting with no knowledge, the algorithm learns to make beneficial routing decisions over the life of the ALV.

The ALV has an overarching goal of exploration; it is charged with exploring as much of the terrain as possible. This goal constrains the ALV's movement (in a way to be discussed later), and thus the routing decisions of the algorithm.

The algorithm utilizes two basic processes: pattern matching, including a determination of closeness of match between nonidentical patterns, and weak induction. The algorithm's underlying assumption is that routing decisions that had positive or negative consequences in the past will have similar consequences in similar situations in the future. This assumption is the basis for a form of inductive learning. However, the algorithm does not attempt to construct general rules from specific experiences; rather, it simply relates each situation it encounters to the most similar situation from its past experience.

The question to be answered can be stated as: could this simple algorithm produce learning in the context of the ALV routing problem, within the constraints imposed by the ALV's goal of exploration? The sections of this paper will relate the ALV simulation scenario, the learning algorithm, and the experimental results.

ALV SIMULATION SCENARIO

A simulated autonomous land vehicle is randomly positioned on a piece of simulated terrain that it must explore. For example, the scenario can be thought of as an interplanetary robot probe on the surface of an extraterrestrial planet. In order to explore, the ALV must move about the terrain entirely without outside control. As it moves, it may encounter a variety of terrain types. These terrain types include sheer bluffs which the ALV cannot traverse, rocks or sand that damage the ALV, or small amounts of a mineral that the ALV can use to replenish its fuel supply.

To simplify the simulation, this terrain is discretized into a square array of cells. The ALV can move from cell to cell orthogonally, but not diagonally. Each of the cells has a single terrain type. Based on the terrain type of a cell and the number of consecutive moves the ALV has made into cells of that type, moving into a cell may benefit or damage the ALV.

Benefit or damage is reflected in the ALV's state, an integer in the range -32 to 32. The ALV's state begins at 32, and if it reaches -32, the ALV is disabled. In addition to the benefit or damage incurred by the ALV when moving based on the terrain type of the cell moved into,
the ALV's state is also reduced by one for each cell moved to reflect wear and fuel expenditure.

Admittedly, the use of a single state variable to represent both damage and fuel is simplistic, but not unacceptably so. The state value range of -32 to 32 was chosen arbitrarily, and the terrain effects were scaled to match it.

It must also be noted that when the ALV moves into a fuel cell, that fuel is consumed and the cell converted into a clear cell.

<table>
<thead>
<tr>
<th>Terrain type</th>
<th>Frequency</th>
<th>Consecutive moves in terrain type</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5+</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fuel</td>
<td>150</td>
<td></td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Clear</td>
<td>490</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Bluff</td>
<td>124</td>
<td></td>
<td>-1</td>
<td>-1</td>
<td>-2</td>
<td>-2</td>
<td>-4</td>
</tr>
<tr>
<td>Rocks</td>
<td>80</td>
<td></td>
<td>-1</td>
<td>-2</td>
<td>-4</td>
<td>-8</td>
<td>-16</td>
</tr>
<tr>
<td>Sand</td>
<td>80</td>
<td></td>
<td>-2</td>
<td>-4</td>
<td>-8</td>
<td>-16</td>
<td>-32</td>
</tr>
</tbody>
</table>

Table 1. Terrain types and benefit/damage caused by each type.

```
RRRR RRRR SSSS RRRRRSSSSSSFF
RRRR FFFFFF SS SSSS RRRRRSSSSSSFFF
RRSS RS S RRRR SSSSSSFF
RRSF SS FFFF RR FF FFFF RRRR F
RRRFS SSSR RRRRR FFF RRRRFF RR RRRRFS S
RRSS R RRRRR FFFF SSSSS SSRR FF SSSS FFFFFFFF RRRRFS SS SSSS FF
RFFF R FF RFFF R FF RFFF RRRRRFF RRRRRFF
RRRR RRRRR RRRRR FFFF SSSSS SSSS FFFFF RRRRR FF
RFFF FF R RRRRR FFFF SSSSS SSSS FRRRFF RRRRRFF
```

Figure 1. Terrain layout.

The ALV's goal of exploration requires the ALV to move across the terrain. From any given cell, the ALV would normally be able to move into any one of the four orthogonally adjacent cells. However, if the ALV's state is > 0, its move choices are restricted to the subset of those four moves that take the ALV in the direction of unexplored areas of the terrain. Therefore, the ALV can not avoid damage simply by circling or moving aimlessly in areas of safe terrain. It must move into new terrain cells. However, if the ALV's state falls below 0, it is permitted to move in any direction until it has raised its state above 0 by finding fuel.

The ALV is equipped with sensory receptors that can distinguish one terrain type from another, but it does not have any a priori knowledge as to which terrain types are beneficial and which are harmful. Since we wanted to study learning from a starting point of no knowledge, the ALV is initially information poor. Thus its early movements will be random, and it must learn by trial and reinforcement which terrain types move into and which to avoid.

Because of the exploration requirement, the overall hostility of the environment, and the expenditure of fuel while moving, the ALV will inevitably be disabled (entropy increases!). The ALV's objective is to remain operational as long as possible and to explore as much terrain as possible. The effectiveness of the learning algorithms controlling the ALV will be measured in terms of number of moves before the ALV is disabled (its 'lifespan').

**LEARNING ALGORITHM**

The learning algorithm that controls the ALV is divided into two components. The first component selects each move for the ALV by comparing the current situation to the ALV's past experiences. The second component learns about the selected move after it is made based on the move's consequences. This is the familiar 'generate and test' structure for a learning algorithm, identified in [Simon, 1981] and described in [Forsyth, 1989] as a unifying principle of machine learning.

The algorithm uses a pattern matching technique to select a move. The ALV moves through the terrain in discrete steps. Before each move, the terrain types of the cells which the ALV can reach in two moves are assembled into a string or pattern. This pattern represents the ALV's current situation.

A table, called the experience table, contains all of the patterns that the ALV has previously encountered (the experience table is empty at the start of a trial). The algorithm searches the experience table for the pattern in the table that most closely matches the current pattern. When evaluating the degree to which two patterns match, the closest match determination considers the magnitude of the difference in terrain types between two patterns, and more heavily weights agreement/disagreement in pattern positions corresponding to cells adjacent to the ALV (i.e. that it may move into on its next move).
Associated with each pattern in the experience table are four move ratings, one for each of the ALV's possible moves. It is by adjusting these ratings in a manner to be described below that the algorithm learns. The highest rated move is selected, with ties broken arbitrarily. This process is illustrated in Figure 2.

\[
\begin{array}{cccc}
3 & 3 & 3 & 2 \\
4 & 4 & 2 & 2 \\
4 & 2 & 2 & 1 \\
5 & 5 & 2 & 1 \\
5 & 5 & 4 & 2 \\
\end{array}
\]

5x5 sub-array of terrain array, centered on the ALV. Terrain types are represented as integers.

\[
\begin{array}{cccc}
9 & 9 & 9 & 9 \\
8 & 1 & 5 & 5 \\
12 & 4 & 2 & 10 \\
7 & 3 & 6 & 6 \\
11 & 11 & 11 & 11 \\
\end{array}
\]

Template for encoding the current situation as a pattern. Entries indicate the order in which cells are represented in the pattern.

\[
\begin{array}{cccc}
22522543244 & 22522543244 & 22522543244 & \ldots \\
\end{array}
\]

Pattern representing current situation.

\[
\begin{array}{cccc}
22522543244 & +1 & -2 & 0 -4 \\
\ldots & \ldots & \ldots & \ldots \\
\end{array}
\]

Pattern Experience table

Ratings

Figure 2. Move selection process.

In Figure 2, the current situation exactly matched a pattern in the experience table. Often there will be no exact match, especially at the beginning of a trial. When there is no exact match, the algorithm finds the pattern in the experience table that most closely matches the current pattern, and uses its ratings to select a move. It also adds the new pattern to the experience table and sets the new pattern's associated move ratings to the move ratings of the most closely matching pattern. Thus, as the algorithm begins to learn about a new situation, it uses its knowledge of the most similar situation previously encountered as a starting point.

When the selected move is executed, it will produce an adjustment to the ALV's state, which is the sum of the cost (or reward) associated with the terrain type moved into (see Table 1) and the fuel expended for the move. The consequence of the selected move is added to the move rating for the appropriate pattern and move. Beneficial moves increase the rating, and harmful moves decrease it. Through this mechanism, the algorithm learns which moves to make and which to avoid.

The algorithm conforms to the basic paradigm of a learning automata defined in [Narendra, 1989], in that it selects an action from a set of possible actions, and then adjusts the probability of that action being selected again based on the consequences of that action. An early and very simple example of such an automata is described in [Gardner, 1962].

EXPERIMENTS

Two different experiments, each consisting of a number of trials, were conducted using the ALV simulation and the learning algorithm previously described. This section will report the results of those experiments.

In the first experiment, the performance of the learning algorithm was compared to the performance of an algorithm that chose the ALV's moves at random. The intent was to determine if the simple learning algorithm did in fact produce improved performance through learning.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Random</th>
<th>Weak inductive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of trials</td>
<td>250</td>
<td>250</td>
</tr>
<tr>
<td>Shortest lifespan</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>Longest lifespan</td>
<td>96</td>
<td>138</td>
</tr>
<tr>
<td>Mean lifespan</td>
<td>50.1</td>
<td>64.7</td>
</tr>
</tbody>
</table>

Table 2. Experiment 1 results.

To show that learning was taking place, the lifespan distributions of the two algorithms were compared statistically. Because the data could not be assumed to have a normal distribution, a non-parametric (i.e. distribution-free) hypothesis test was needed. We used the Sign test, as defined in [Bhattacharya, 1977], with a 95% confidence level.

\[ H_0: \] the lifespans of the two algorithms are the same

\[ H_1: \] the weakly induction algorithm produces larger lifespans

The value of the test statistic \( S = 168 \), and the number of trials \( n = 250 \) (there were no ties in the pairs). In the Sign test, the test statistic \( S \) has a binomial distribution, but because of the large sample size, we used the normal approximation to the binomial distribution. For the large sample approximation, the value of the test
Figure 3. Experiment 1 lifespan frequencies.

Figure 4. Experiment 2, average state by clock (50 trials).

The statistic $z$ is calculated as:

$$z = \frac{S - \bar{X}}{\frac{S}{\sqrt{n}} = \frac{163 - \frac{250}{2}}{\sqrt{\frac{260}{4}}} \approx 5.44$$

The value needed to reject $H_0$ in a one-tailed test with a 95% confidence level is 1.64. Because $z = 5.44 > 1.64$, $H_0$ is rejected and $H_1$ accepted. Therefore, the weakly inductive learning algorithm does produce verifiable learning.

A limited attempt to generalize the reinforcement scheme, so that the last $i$ moves ($i \geq 2$) are reinforced, did not produce measurable improvement in our tests.

For the second experiment we used the original weakly inductive algorithm, but removed the lower limit on the ALV's state. That is, the ALV was given infinite fuel and the capacity to absorb infinite damage. Another way of interpreting this is to think of a consecutive series of ALVs, each one starting with the experience table of its disabled predecessor. The lifespan of the ALV was limited arbitrarily to 500 moves. Here, we wanted to determine what the long term performance of the algorithm would be. How close could it come to optimum behavior?
A benchmark state was defined to be used as a standard for comparison. The ALV starts at state 32 and without considering terrain effects, decrements that state by one each move due to fuel expenditure. Thus the benchmark state value = (32 - number of moves made). A state value above the benchmark indicates that the algorithm is avoiding damage and finding fuel well enough to more than compensate for the cost of movement. Below the benchmark, the algorithm is not avoiding damage or finding fuel effectively.

As can be seen in Figure 4, which averages the results of 50 trials, the weakly inductive algorithm would typically go through a period during which its lack of a priori knowledge caused it to perform below the benchmark. Then it would surpass the benchmark for a longer period, before sinking below it again. Observation of the trials revealed that the fall off at the end of the trials was due to the exhaustion of the fuel deposits in the terrain.

SUMMARY AND CONCLUSIONS

In this paper we presented a simple learning algorithm and reported its performance in routing a simulated autonomous land vehicle. The algorithm has no initial knowledge, and uses two well-known learning mechanisms, pattern matching and induction.

Our experiments showed that the algorithm does produce measurable learning. Over the long term, the algorithm learns to produce nearly optimum behavior in its domain.

We speculate that some possible enhancements to the algorithm that have already been identified (e.g. considering rotational equivalence when searching for the closest matching pattern) would significantly improve the algorithm's performance. Finally, ([Dotterweich,1989] showed in work subsequent to our experiments that a genetic algorithm system was effective on a very similar problem.

REFERENCES


EVALUATION OF THE LEARNING PROCESS IN A
SELF-ADAPTIVE EXPERT SYSTEM

David R. Berlin and Mordechay Schneider
Department of Computer Science
Florida Institute of Technology
Melbourne, Florida 32901-6988

ABSTRACT

The use of exhaustive search algorithms in highly interactive expert systems is often hampered by long processing delays for each user input. While increasingly powerful processor hardware continues to appear in the computer market, these advances are more than offset by user expectations for faster results to more complex problems. There is also a wide range of real-time applications which are well suited to utilize rule based expert system techniques provided the inference engine could respond in a timely manner.

This paper describes the methodology used in evaluating a learning function for rule based expert systems. The learning function being evaluated is designed to optimize the application of existing rules rather than create new rules. This type of learning helps reduce the time required to process input to expert systems which have an extensive knowledge base.

INTRODUCTION

The task of optimizing the performance of a rule based expert system may be divided into two classes. On the one hand there is static optimization which is performed as the rule base is created. In this case the inference process is simplified by a rule base preprocessing stage. Another approach, dynamic optimization, is performed as the expert system executes. This type of optimization reduces processing overhead by adjusting the order in which rules are exercised. Learning to solve problems quicker is the result of dynamic optimization.

The process of determining the goal state values for each user input involves a chain reaction of rule firing. Each time a clause is set to a new value, the rules containing that clause are applied. If, on the other hand, the rule does not result in a new value, then the current branch is terminated. When multiple paths exist between two clauses, a single input can cause the down stream clause to be evaluated twice. If the value which was assigned first is superseded, the entire branch below that clause is re-evaluated. The redundant processing of the rule base is minimized when the rules which assign the final value to each clause are executed first. In order to choose the preferred path, an execution priority for the rules must be established. This priority should not affect which rules are executed but should determine the order in which they are executed. The assignment of this execution priority is the dynamic optimization which results in the system learning to solve problems quicker.

The Dense Rule Format

As the rule base is entered, the rules are parsed into clauses. The clauses are matched to existing clauses. The clauses are stored in a list of records which are indexed by a numeric clause identifier (CID). If a new clause is encountered, it is added to the existing set. The clause records contain descriptive information which allows the user interface to associate them with information provided in sentence structure and to build queries when the system must request information. As the clause matching and creation process proceeds, the rules are rewritten in the dense internal format. In this format the text of each clause is replaced by its CID and numeric codes represent the brackets and logical operators. The rule storage is structured so that the rules can be referenced by the numeric rule identifier (RID). Once the rules have been copied into the dense internal format by the human interface component, a list of RIDs is produced which is used to create the rule selection construct. Initially the RID in each list element matches the index which points to it. This list will be rearranged by the learning function when the expert system is used.

The Clause to Rule Cross-Reference

Once the dense format rule base has been created the construct used to select rules for execution is produced. The structure used is an array of records where
each record contains a RID and a pointer to the next record in the list. The head of each list is pointed to by the CID and the last record in each list contains an specific END-OF-LIST value in the linkage pointer.

In order to create the lists, one record for each clause is initialized to contain a null RID and the END-OF-LIST pointer. These records serve as list heads. The dense format rules are then scanned in the order specified by the learning function. Each time a CID is encountered in the rule premise the RID is appended to the list specified by the CID. This process is repeated whenever the learning function reorder s the rule list. When inference process uses these lists to select the rules for execution they will be found in the prioritized order.

Clause Value Propagation

Once the rule base is compiled, the expert system is ready to begin solving problems. The inference engine uses the cross-reference, along with a stack, to direct the rule firing process. A table of clause state records is used to keep track of the current value of the clauses and which rules determined their value. Related clauses are evaluated by a depth-first rule firing algorithm described below.

When a new value is assigned to any clause, the recursive routine which propagates the new value is invoked. The recursive nature of this routine causes a record of the current clause and rule to be pushed onto the stack. Both current clause and rule are null for the user evaluated clause. The clause which has been evaluated becomes the current clause and the first rule with the current rule in its premise becomes the current rule. The current rule is fired and if its conclusion is evaluated, the process continues with another recursion. If the conclusion is not evaluated, the next rule containing the current clause is selected using the cross reference. To select the next rule, the linkage pointer of the current cross reference record is loaded into the index which specifies the current record. When all the rules associated with a clause have been processed, the previous clause and rule are popped off the stack by returning from the recursive routine. The next rule is selected from the cross reference list for the restored clause and processing continues.

Analysis of Rule Application

The purpose of the learning function is to reduce the average time required to determine the value of each goal state. This is accomplished by ordering the rules so that the cross reference yields the most efficient rule firing pattern. The optimal configuration for a particular set of rules will vary with the environment in which it is used. Since optimizing the cross reference for one set of inputs may increase the processing time for others, the system should be tuned for the inputs most often provided.

In order to ensure that the rules are fired in their optimal order, the rules are ranked depending on how often they determine the final value for a clause on the path between a goal state and a user input. Each rule has a counter which is incremented at the end of every session if the rule determined the final value of a clause leading to a goal state. This counter determines the RID assigned to each rule and therefore the order in which they are selected using the cross reference.

When a rule is found in the trace, the counter for that rule is incremented. If the rule is found in the trace more than once, it is incremented each time. After the usage counters have been updated the rule list is ordered according to their counters. Rules which have the highest counter values migrate to the head of the list. In the prototype program a bubble up sorting algorithm was chosen because of its simplicity. Actual applications may use a more sophisticated method. In this manner, the rule which is most often successful will be the first expanded when a contributing clause is evaluated, thus reducing the number of paths searched.
EVALUATION METHODOLOGY

Prior to evaluating the effectiveness of the learning algorithm, tests were performed to ensure that the system would execute as required. The first level of testing verified the operation of each routine. The system as a whole was then tested with a sample rule base to ensure that for a given input the resulting goal state values are correct. Once it was established that the system would perform the desired functions, it was possible to proceed with evaluation of the learning function.

Test Objectives

There are three aspects of the learning function being evaluated by these tests. One of the fundamental requirements of the learning algorithm being implemented is that the goal state values resulting from a given input must not be altered by the rule base optimization. This requirement is based on the principle that unless the learning function is proven to consistently improve results, any change in results must be assumed to be detrimental. The second requirement being tested is to achieve an overall reduction of the processing which stems from user inputs. The third feature of the learning function is the ability to find the optimal rule configuration for various user environments. Since the typical value for inputs which determine rule preference may vary the expert system should be able to adapt accordingly.

Test Methodology

Evaluation of the learning algorithm is based on a series of input scenarios which are fed into a sample rule base. The same set of scenarios is run three times. The first run is executed with the learning function disabled. This establishes a baseline against which performance in the optimal configuration can be measured. The second run is executed with the learning function enabled so that the optimization can occur. The third run is executed with the learning function disabled again. This final set of data is used to evaluate performance in an optimized but static configuration. In order to establish the ability of the learning function to adapt to other user environments, another set of scenarios is used. The second set of scenarios represents an environment in which the typical values of user inputs differ from those in the first set.

The Inference Engine Driver

In order to evaluate the performance of the learning function it was necessary to observe the operation of an expert system which applies the function. Rather than develop a complete expert system, the logic unit was interfaced with an inference engine driver. The driver program reads scenario files and presents the input to the logic unit as though received from a human interface. This driver also appends the results of each scenario to the contents of the log file.

Development of a Sample Rule Base

The sample rule base used in these tests is purely abstract. Rather than analyze a real application and create a text format rule, a dense format rule base was developed which includes the multiple path situation. A program was written to compile and store the rule base in a dense rule format file. This file is loaded by the rule base manager when the driver program is invoked. While the sample rule-base is much smaller than a typical expert system would require, it is adequate to demonstrate the operation of the learning function.

Development of the Input Scenarios

Two tools were developed to build the scenario file. The first program produces a stream of random values for each input clause. The second program reads the streams and formats input records which are stored in scenario files. An even distribution random number generator was used with a separate range specified for each input clause. The ranges overlap for inputs which determine the alternate paths between clauses but none is a subset of another. This creates a situation where no path is always the ideal one, but one path is the most likely choice.

Collection of Scenario Results

The measure of processing used in this evaluation is the number of logical operations performed. In order to gather this data, a counter was introduced to the logic unit. Each time a logical operation is performed, the counter is incremented. When the scenario is completed, this counter is stored in a scenario log file. The final goal state values for each input scenario are stored along with the logic operation counter. This data is required in order to ensure that the final results on the input are not altered by the optimization. When all the scenarios in the set have been run, the log file is saved for further processing.

Once the set of input scenarios has been executed as described above, the logic counters are totaled for each pass. These are compared in order to determine how effective the learning function is in reducing the overall amount of processing required. Comparing the totals for the second and third passes will show whether the learning function suffers from thrashing. The logic counters for a particular scenario can be used to study how the input values affect the rule processing. The final goal states are also compared to ensure that each scenario produces the same result in all three passes.

Comparison of Results Before and After Learning

The logic counters from all three passes for each scenario are printed in columns, the goal clause states are compared to ensure that they match, and the logic counters are totaled. It is to be expected that the first set of results in which the learning function has been disabled would have the largest total counter value.
second set of results should show some improvement as the rule base is optimized. The third set of results should be slightly better than the second since the optimal configuration has been implemented for that set of input scenarios.

When the second set of input scenarios is invoked the system will have to adapt to the new user environment. Since this test, unlike with the first set of scenarios, is applied to a system with prior experience a number of consultations are required to establish a trend. Once the trend is established a marked improvement if system response to these scenarios is observed.

RESULTS PERFORMANCE EVALUATION

The results of this evaluation show that the learning function performs as expected. The logic counter totals for each run of the scenarios are illustrated in the graph and table below.

The methods and tools described were used to establish that the learning function described is capable of improving the performance of expert systems which incorporate it. The tests relied on inefficiencies built into the rule base and input scenarios which demonstrate them. This contrivance is justified if only because such problems could arise. The degree of improvement which the learning function will provide in actual expert systems will be determined once it is applied in a number of diverse systems. Further tests are being conducted to establish the scope of improvement which is possible and the factors which determine the actual degree of improvement.

REFERENCES


<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scenario set 1, learning disabled</td>
<td>9364 ops</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scenario set 1, learning enabled</td>
<td>7420 ops</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scenario set 1, learning disabled</td>
<td>7392 ops</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scenario set 2, learning disabled</td>
<td>6039 ops</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scenario set 2, learning enabled</td>
<td>5886 ops</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scenario set 2, learning enabled</td>
<td>5148 ops</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Artificial Neural Networks for Qualitative Reasoning in Design

John H. Garceon and Gale E. Nevill, Jr.

Department of Aerospace Engineering,
Mechanics and Engineering Science
University of Florida, Gainesville, FL 32611

1. ABSTRACT

Analysis and evaluation of preliminary designs plays a fundamental role in the generation of good designs. Concepts from neural network research offer a new and promising approach for the application of computers to this task. This paper gives an overview of preliminary design for and examines the application of artificial neural networks for qualitative reasoning in the field of preliminary structural design.

2. INTRODUCTION

Preliminary design may be characterized as a transformation of functional and other requirements into the general description of an artifact that meets those requirements. The function of the artifact, therefore, defines the form, and the designer uses the functional requirements to generate and guide the preliminary design process. The functional requirements in preliminary design are often given in terms of physical relationships that describe the performance of the artifact.

Analysis provides an evaluation of design performance and guides design decisions. In automated implementations of the preliminary design process, analysis is commonly done using heuristics [6] or by decomposing the problem into "pre-evaluated" subproblems [4]. Although there has been some success using these two approaches, neither is as robust and flexible as human designers.

Although numerical analysis techniques are generally well understood, they require details not available during the preliminary design stage. Human designers evaluate preliminary designs based on a subjective assessment of what they perceive to be the relevant criteria. This seems to be dependent on the human designer's experience level, and designers do this in a purely qualitative way without resorting to numeric simulation. In order to make current design process models more robust, there is an urgent need to develop improved capabilities for performing qualitative evaluations during preliminary design.

2.1 Qualitative Reasoning

Qualitative reasoning of a design artifact typically determines how different variables change in relation to changes in other variables without using specific numeric values. The reasoning is based on cause and effect connections between design features. Qualitative knowledge originates in first principles, such as static equilibrium or constitutive laws, which are fundamental to a design problem domain.

Qualitative analysis of a continuous domain requires that relations between typical design variables map into a discrete qualitative set [2]. The simplest of these discrete sets includes only increasing, decreasing and stationary (+, -, 0) relations, and the reasoning is about the kinds of changes that can occur based on physical laws.

In design problems, physical laws may express a qualitative relationship between features of the design description and can be viewed as constraints among these parameters [3]. Several design researchers have shown the merits of qualitative analysis, but their focus has been on the detailed, parametric design phase [1].

It is clear that qualitative reasoning plays an important role in preliminary design. Unfortunately, to date, there has been only limited success in utilizing computers for this task. Artificial neural networks provide a promising new approach to constraint satisfaction problems and thus qualitative reasoning.

3. ARTIFICIAL NEURAL NETWORKS

Traditional neural network research areas, such as pattern matching and vision processing, have a number of characteristics in common with design models. They are often described as constraint satisfaction problems, where a large number of constraints must be satisfied simultaneously and where input may be incomplete or inconsistent. Artificial neural networks (connectionist systems) are well suited to solving these types of problems.

Given a preliminary design problem, there commonly exist a huge number of feasible, valid designs that meet minimal functional specifications. If the requirements and choices are able to simultaneously and mutually constrain and guide one another, then there is an increased possibility that "good" design candidates will result. Such a simultaneous system explores the alternatives without committing to any particular one until all the constraints are taken into account. Connectionism naturally supports simul-
aneous consideration of multiple constraints and thus intrinsically and efficiently exploits a "least commitment" problem solving strategy.

3.1 "Thermodynamic" Network Models

A class of models well suited to qualitative reasoning extends basic network characteristics by using concepts from thermodynamics to find stable network states. The following examination of the connectionist approach to constraint satisfaction problems using thermodynamic models follows Smolensky's work on Harmony Theory [7, 8].

Constraint satisfaction problems are posed to connectionist models such that each processor (or group of processors) represents a possible solution value for a variable, and connections between processors and groups of processors represent constraints on relations between variables. Connections can either be positive (for supporting states) or negative (for inhibitory states.)

Starting conditions are given to a network and activate appropriate processors for that state. Those activations are then propagated through the network. The problem is solved when the activations of a particular set of processors are maximized. The network does this by maximizing the degree of constraint satisfaction of each processor.

For a processor to maximize its degree of constraint satisfaction, it only needs to examine the sign of its total input, x. If x is positive, the processor increases its activation a; if x is negative, the processor decreases a. Thus, each processor mutually influences those processors connected to it such that the system as a whole tends to maximize the total degree of constraint satisfaction. The total degree of constraint satisfaction increases only until all processors are maximally active or inactive, thus the network achieves a stable state.

This method is essentially a hill-climbing optimization mechanism and can only be guaranteed to find local optimum solution states. In order to overcome local optima, thermodynamic neural network models were developed.

To prevent a system from settling onto a local optimum, the system is perturbed to encourage it to move away from the plateau of the local optimum. This concept is called simulated annealing. A global parameter, analogous to temperature in physical systems, is included in the stochastic activation function \( p(x) \) (Figure 1). The activations are initially low because the high temperature in the system flattens the activation function. As the system processing continues, the temperature is lowered, allowing the system to "cool." The activations gradually settle onto a stable global optimum state.

In the function given in Figure 1, the activation probability is plotted versus the total input to a processor. The plot shows the activation probability of an active processor \( a_i = 1 \) for a calculated input, \( x \). The temperature, \( T \), in this equation determines the range of activation uncertainty and is set by some annealing schedule. When a processor satisfies all its input constraints, then its activation will always be 1, regardless of the temperature.

Harmony theory [8] is one of several specific thermodynamic connectionist systems and is chosen as representative of the system proposed to date. Harmony theory specifies two distinct layers of processors, knowledge atoms and representational features. Its basic concept is that for any set of input representational features, the system will find a set of knowledge atoms that are harmonious with the given features.

![Representational Features](image)

**Figure 2: Harmony Network**

Conventional knowledge structures are in the form of frames and scripts, which are normally fixed symbolic descriptions. In contrast, harmony theory does not presuppose a particular unchanging environmental depletion; it dynamically activates the knowledge atoms based on the active representational features. Figure 2 depicts part of a graphical representation of a harmony network for several qualitative relationships, showing the two levels and constituent knowledge atoms and representational features. Two basic topological properties should be noted. First, processors are only connected between levels (not within levels), and second, the connections between processors are symmetric.

For a given harmony network, a stable state is found by fixing some of the representational features based on the network's input. Then, each of the processors continually updates its value according to the stochastic activation function and the current harmony. Each processor computes its harmony using as input the harmony of all
its connected processors. A processor’s probability of moving towards lower harmony is greater for higher computational temperatures. Alternately, all the knowledge atoms update, and then all the representational features update. By alternating between levels, harmony networks fill in missing information in a consistent manner. Therefore, given an incomplete problem description, harmony models "interpret" the situation and yield a maximally consistent state, satisfying the constraints to the extent possible.

The harmony of each knowledge processor, $h_i$, is a measure of how consistent the processor is with respect to its connected feature processors. Each knowledge processor's harmony function is given by:

$$ h_i = \frac{\sum w_j y_j}{n_i} - \kappa $$  \hspace{1cm} (1)

The connections between processors, $w_j$, take on trinary values (-1, 0, 1); $y_j$ is the activation of representational feature $j$; $n_i$ is a normalizing factor, the number of non-zero connections to processor $i$. The parameter $\kappa$ acts as a threshold constant and regulates what proportion of the processor's feature input ($y_j$) must support the knowledge before knowledge activation occurs. When $\kappa = -1$, the above equation does not require any of the representational features to activate the knowledge atom. When $\kappa = 1$, the equation requires all connected features to be present before activation. Since there is a limit to how close to an exact match one can get with a finite number of possible matches, $\kappa$ specifies what constitutes an acceptable match for $h_i$.

The harmony of the total system is the sum of the harmonies of all knowledge atoms and is given as:

$$ \text{harmony} = \sum a_i a_i h_i $$  \hspace{1cm} (2)

Each knowledge atom's activation, $a_i$, takes on binary values (0, 1) based on the probability function. The bias term, $a_i$, signifies a knowledge atom's importance based on some a priori belief.

The following example illustrates the formulation of qualitative analysis tasks in terms of artificial neural networks and describes network performance.

**4. EXAMPLE**

The following example is taken from preliminary structural design. It is assumed that specific requirements are given in the design problem description, and the network is asked to qualitatively evaluate the design problem in order to determine the best way to meet those requirements. This example describes the formulation and behavior of a harmony model and uses computer code originally obtained from [5].

![Figure 3: Abstract Structure](image)

A typical structural design problem may be stated as: "Design a stable structure, to support given loadings, such that the displacements are small and the stresses are within allowable ranges." Figure 3 shows what an abstraction of such a design problem could look like. The following simultaneous equations represent the force equilibrium relationship:

$$ F_1 = (k_1 + k_2) * u_1 - k_3 * u_2 $$  \hspace{1cm} (3)

$$ F_2 = -k_3 * u_1 + (k_2 + k_3) * u_2 $$  \hspace{1cm} (4)

In the above form, these equations are difficult to represent qualitatively; therefore, each equation is decomposed such that equation (3) becomes:

$$ S_9 = k_2 + k_3 $$  \hspace{1cm} (5)

$$ S_2 = k_3 * u_2 $$  \hspace{1cm} (6)

$$ S_1 = k_1 + k_2 $$  \hspace{1cm} (7)

$$ F_2 = S_2 - S_1 $$  \hspace{1cm} (8)

In a similar manner, equation (4) is decomposed into:

$$ S_2 = k_2 * u_2 $$  \hspace{1cm} (9)

$$ S_2 = k_2 + k_3 $$  \hspace{1cm} (10)

$$ S_1 = S_2 * u_2 $$  \hspace{1cm} (11)

$$ F_2 = S_2 - S_1 $$  \hspace{1cm} (12)

A simple constitutive relationships provides knowledge relating displacements to stresses:

$$ f_1 = E * u_1 $$  \hspace{1cm} (13)

$$ f_2 = E * (u_2 - u_1) $$  \hspace{1cm} (14)

Equation (14) is decomposed into:

$$ d = u_2 - u_1 $$  \hspace{1cm} (15)

$$ f_2 = E * d $$  \hspace{1cm} (16)

Any other related equations may also be incorporated in the qualitative analysis. More equations further constrain the solution and usually make solutions easier to identify, provided sufficient design specifications are given for the equations. The following abstract stiffness equations are therefore added:

$$ k_1 = E * A_1 $$  \hspace{1cm} (17)

$$ k_2 = E * A_2 $$  \hspace{18}

$$ k_3 = E * A_3 $$  \hspace{18}

All the above equations are a basis for qualitative relationships among important abstract design variables: stiffness $k$, displacement $u$, external force $F$, elastic modulus $E$, cross-sectional area $A$, and stress $f$. Several temporary variables, $S$, allow for the decomposition of the more
complicated equations into simpler ones.

To use harmony theory, a set of representational features and knowledge atoms must be defined. The representational features should embody the requisite qualitative facets. The above equations describing the abstract structure provide the basis for a set of twenty-one representational features \( k_1, k_2, k_3, u_1, u_2, F_1, F_2, E, f_1, f_2, f_3, A_1, A_2, A_3, S_1, S_2, S_3, S_4, S_5, d \). The behavior of these features need to be qualitatively represented as increasing (+), decreasing (-), or unchanging (0); therefore, each representational feature consists of two neural processors, one indicating the presence of change, the other indicating direction as either increasing or decreasing. Thus, the harmony model for this system requires a total of forty-two representational processors.

Knowledge atoms represent knowledge about the domain: equilibrium, constitutive, and stiffness equations. All possible relationships between features in these equations must be determined and represented by a processor. For example, equation (18) can describe a number of different relations between parameters. If \( A_1 \) and \( E \) increase, then \( k_1 \) must also increase. Similarly, if \( A_1 \) increases and \( E \) is unchanging, then \( k_1 \) will increase. If \( A_1 \) increases and \( E \) decreases, then \( k_1 \) can increase, decrease or remain unaltered. All possible valid combinations need to be encoded into the network. For each decomposed equation, there are thirteen legitimate qualitative relationships. With fifteen equations, this leads to 195 knowledge processors. The connections \( (w) \) between the knowledge and feature levels symbolize the qualitative constraints that comprise a portion of a valid abstract design state (see figure 2).

indicating no a priori constraint knowledge.

With the network formulated, it is now possible to perform a qualitative analysis of this structural system. The input to this network signifies functional design requirements and is as follows:

- The applied forces \( F_1 \) and \( F_2 \) are expected to be unchanging \( (F_1', F_2') \).
- A single material will be used so the elastic modulus is unchanging \( (E') \).
- The displacements \( u_1 \) and \( u_2 \) should be small, so indicate them as decreasing \( (u_1', u_2') \).
- The stresses \( f_1, f_2 \) and \( f_3 \) should be small, so indicate them as decreasing \( (f_1', f_2', f_3') \).

Given this input, the network updates the activations of each processor in a random order, once per cycle. Providing that the given problem is well posed, the network activates consistent constraints. Figure 5 shows the system’s harmony value, specified in equation (2), as it changes for each cycle. As can be seen, the harmony asymptotically approaches a value of 4.0 as the system is “cooled.” This indicates that the system is settling onto a maximum. The active constraints on the design variables after 300 cycles are: \( k_1', k_2', k_3', A_1', A_2', A_3', F_1', F_2', E' \) \( u_1', u_2', f_1', f_2', f_3' \), which is consistent with the input specifications and the possible valid outcomes.

Figure 5: Harmony Values

Interesting network behavior and challenging qualitative analysis is seen when a problem is posed in an ambiguous or even contradictory manner. The previous problem is revised to create an ambiguous problem and presented to the network as:

- Make the displacements \( u_1 \) and \( u_2 \) small \( (u_1', u_2') \).
- Make the stresses \( f_1, f_2 \) and \( f_3 \) small \( (f_1', f_2', f_3') \).

This is ambiguous since the input specifications allow for multiple valid scenarios where many qualitative relationships suffice. The input features do not assist in constraining all other processors, making a single harmonious result more difficult to find. Using the same annealing schedule, the system’s resulting harmony function values are similar to those in figure 5. After 200 cycles with a system harmony of 4.0, the following design variable constraints were

---

\[ \text{By having all processors use binary activations the network implementation is simplified.} \]

\[ \text{The proportionality constant is determined from: } 1 > k > 1 - \frac{1}{n}. \text{ Three features are assigned with each knowledge atom and two processors represent each feature; therefore, } n = 6, \text{ which yields: } 1 > k > 0.67. \text{ When } n = 6, \text{ [8] suggests using } k = 0.75. \]
active: \(k^* \); \(k^* \); \(k^* \); \(A^* \); \(A^* \); \(A^* \); \(F^* \); \(F^* \); \(E^* \); \(u^* \); \(u^* \); \(f^* \); \(f^* \). However, \(A^* \) indicates that the system has not yet decided upon which knowledge atom and constraint set should be activated to specify \(A^* \). Likewise, the system has activated several knowledge atoms that contradict one another. These are variations of equation 16: 
\[
d^* + u^* = u^* \quad \text{and} \quad d^* + u^* = u^* \quad \text{and equation 17:}
\]
\[
d^* + E = f^* \quad \text{and} \quad d^* + E = f^* \]
This result is consistent with respect to the input specification, but the network activated two sets of conflicting constraints on the temporary variable \(d^* \), which is inconsistent. At this stage of the processing, the network "could not make up its mind" regarding which constraint is best; however, both are consistent when taken individually, and both support the equilibrium, stiffness and constitutive constraints.

After 300 cycles with a system harmony of 4.167, the following design variable constraints were active: \(k^* \); \(k^* \); \(k^* \); \(A^* \); \(A^* \); \(A^* \); \(F^* \); \(F^* \); \(E^* \); \(u^* \); \(u^* \); \(f^* \); \(f^* \); \(f^* \)

Again, several contradictory knowledge atoms were activated for equations (10, 16 and 17); however, the system has specified a consistent constraint on \(A^* \). Several equilibrium and stiffness constraints changed from the 200 cycle result, but all constraints are still consistent with respect to the input features and knowledge atoms. In addition, the system harmony value of 4.167 is above the ideal global maximum of 4.0 as given by equation 2. This is due to multiple knowledge atoms positively contributing to the harmony even though both can not occur simultaneously. With more initial specifications as in the first run, the network performs with more "decisiveness."

Analysis of these networks indicates that due to the simultaneous equations, the number of processors and connections required increases as a polynomial function; however, the run time necessary to find an answer is exponential with respect to the number of processors and connections. This is due to the simulation of a parallel distributed process on a serial machine.

5. DISCUSSION

Constraint satisfaction models appear well suited to support automated preliminary design. They deal efficiently with numerous constraints and can develop consistent results given conflicting or incomplete requirements. These properties of connectionist systems, as shown in the above examples, indicate that such systems offer researchers a promising, new tool for use in automating preliminary design.

They are not without costs, however. This tool requires an interface with existing automated design systems, in order to utilize requisite abstractions for creating the network. Automating both the abstraction process and network topology construction are both necessary components of an effective design tool. There may be a small number of abstractions and associated network topologies that adequately characterize a large number of structural preliminary design situations. In this case, an automated design system could then initiate a qualitative evaluation using appropriate abstractions and networks, obtained with acceptable effort. This is an obvious direction for further research.

Networks do not recognize when convergence has been achieved, nor do they identify ambiguous input and results. This robustness is an inherent property of connectionist systems; however, for some purposes, reasoning should produce unambiguous results. At other times, ambiguity in the evaluation could be used to advantage, for example, by suggesting multiple approaches. Regardless, the network needs to indicate when an ambiguous result occurs. In this paper's model, convergence and ambiguity are identified by human observation. Overall, it seems clear that artificial neural networks offer a promising capability, complimentary to current approaches for qualitative reasoning.

6. ACKNOWLEDGEMENT

The work reported was supported by the National Science Foundation under Grant No. DMC-8813808.

7. REFERENCES


FRAME PLANNER

Frank M. Brown, Salwinder S. Hundal and Carlos L. Araya
Artificial Intelligence Laboratory
Department of Computer Science
The University of Kansas
Lawrence, Kansas, 66045
allab@csvax.cs.ukans.edu
(913)864-4482

0. ABSTRACT

A theory of planning that uses nonmonotonic reasoning based on the modal logic Z (Brown 1986a) is developed. It does forward reasoning and backward planning to reach the goal. The theory we are proposing uses the frame axiom and the modal logic Z to propagate the facts from the current situation to the next situation. The explicit results of an action are provided only, no delete list is needed. The facts are automatically added and deleted from one situation to the next by the nonmonotonic reasoning as the actions are performed. The planner called "The Frame Planner" is written in the Deduction Language Schemata developed in our lab, currently running on Symbolics and Macintosh machines.

1. INTRODUCTION

Planning consists in constructing a sequence of actions that leads from a given initial state to a final state to accomplish given goals. STRIPS (Fikes et al. 1971) like traditional planners do not address the frame problem (Hayes 1973) because their mechanism of carrying facts from one state of the world to the next state is all in the descriptions of the actions, which is cumbersome for the user and many times it is impossible to describe all the effects of an action on the existing state of the world. In this paper a theory of planning with nonmonotonic reasoning using the modal logic Z (Brown 1987a) is presented. In this theory, only the explicit results of an action are provided and no delete list is needed. The facts from the current situation to the next situation are carried out by the frame axiom (Brown et al. 1987b), and the facts are automatically added and deleted from one situation to the next by the nonmonotonic reasoning as the actions are performed. The reasoning mechanism is forward but the plan is constructed backward.

The Frame Planner is written in the Deduction Language Schemata and the nonmonotonic modal logic Z systems (Brown 1989), both are developed in our AI lab, and currently running on the Symbolics machines. Schemata is a language for writing deduction systems described in (Araya et al. 1989). It has very general metalogical capabilities so that a logic system can be defined by simply specifying its axioms and inference rules. In particular, Schemata implements the fundamental deduction principle of SYMBAL (Brown 1986b) as well as an efficient pattern matching and backtracking mechanism. The language adopts some of the powerful features of Scheme (Rees et al. 1986). The nonmonotonic system that uses the modal logic Z is written in Schemata as a subsystem.

2. THE FRAME PLANNER

The frame planner takes the goal, the initial situation and the frame laws as inputs and it returns all possible plans. To get the first plan and delaying the other plans, we can use the join primitive of Schemata which converts multiple solutions of a problem into a stream. The Frame planner has three main components, which are the solve, the action-applier and the actions-generator. The define function of Schemata is similar to the define of Scheme except that in Schemata the variables can have multiple definitions and each definition produces a backtrack point for the execution hence producing multiple solutions corresponding to the successful application of each definition. Consequently, any evaluation of Schemata produces a stream of solutions instead of just one value as in other LISP systems. This peculiarity provides also a natural way to represent failure which is explicitly produced by the return of the $fail$ value. The define function is like the define but with a "cut!" that means the execution of a definition with "cut!" will stop the application of any other definition of the same variable. During execution, the definitions are applied from bottom up (ie.,top/first definition will be applied last) when they match the argument list. For example, the pattern (\textbf{\textless} fact) in the parameter position of the first definition of -filter restricts its application to only those cases in which the same pattern appears in any position of the argument. In this case, \textbf{\textless} is a constant matcher and \textbf{fact} is a variable which will be bound to the corresponding piece of argument during execution. The \textbf{\textbullet\textbullet\textbullet} is a segment variable for pattern matching which produces horizontal searches through the arguments list. The planning theory is given in the following algorithm. The logical syntax and semantics of the algorithm is discussed briefly in section 5 on the quantificational modal logic Z.
(define (frame-planner goal K0 frame-laws)
  (define (solve goal (plan K0) old-bel)
    (if (is-subsumed K0 old-bel) $fail
      (let((actions (actions-generator goal K0)))
        (if.null? actions)
          (list plan K0)
          (solve goal (action-applicator (list plan K0))
            (split-actions)
            (cons K0 old-bel))))))

(define (actions-generator goal K0)
  (sort (lambda (a b) (reduce-diff? (results a) (results b))
    (filter (lambda (a) (eq? (exec-nm (-1 K0) (results a))))
      (find-right-actions goal K0))))

  (define (filter test)())
  (if (filter test (next-action . rest-actions))
    (cons next-action (filter test rest-actions))
    (filter test rest-actions)))

  (define (solve goal (nil K0) nil)) ;end of the function frame

  (define (exec-nm exp
    (if (equal? exp (one fact))
      (+1 exp)
      (exec-nm exp))

  (define (variable K)
    (solve goal(nil K0) nil)))

    (define (solve goal (plan K0) old-bel)
    (if (is-subsumed K0 old-bel) $fail
      (let((actions (actions-generator goal K0)))
        (if.null? actions)
          (list plan K0)
          (solve goal (action-applicator (list plan K0))
            (split-actions)
            (cons K0 old-bel))))))

    (define (actions-generator goal K0)
      (sort (lambda (a b) (reduce-diff? (results a) (results b))
        (filter (lambda (a) (eq? (exec-nm (-1 K0) (results a))))
          (find-right-actions goal K0))))

      (define (filter test)())
      (if (filter test (next-action . rest-actions))
        (cons next-action (filter test rest-actions))
        (filter test rest-actions)))

      (define (solve goal (nil K0) nil)) ;end of the function frame

      (define (exec-nm exp
        (if (equal? exp (one fact))
          (+1 exp)
          (exec-nm exp))

      (define (variable K)
        (solve goal(nil K0) nil)))

The solve function starts with a goal, an empty plan, the current situation and an empty old belief set to generate a stream of plans represented as list of actions. First, it checks if the current situation is subsumed by the old belief set, in that case the search on this branch is called off and the next alternative is explored. Otherwise, it calls the actions generator to find a list of possible actions to solve the goal. If the action list is empty then solve returns the plan and the current situation, else it splits the execution for every possible action. The split function produces the backtrack points in depth first order. When an execution falls the Schemata system truncates that branch and alternative execution is followed. Again, the solver calls itself with the goal, the new situation

that is produced by the application of an action, and the old belief set after adding the current situation into it.

The actions generator function takes a goal and the current situation as input and returns a list of possible actions that are necessary to achieve the goal. It finds all the appropriate actions for achieving the goal, filters those actions which are not necessary and sorts the actions in descending order of their capability to reduce the difference between the current situation and the goal situation (i.e., an action whose result is closer to the goal comes first). The closeness criterion is problem dependent and is provided by the user.

The action applicator transforms the current situation to a new situation after applying the frame law to the result of the action. The frame laws are defined in the problem domain. The action applicator uses nonmonotonic system which is built upon the quantification modal logic 2 (Brown 1989) as described in the Appendix. Before an action is applied to the current situation, the solve function is called to solve the preconditions of the action as subgoals.

3. PROBLEM DEVELOPMENT - THE BLOCKS WORLD - SUSSMAN ANOMALY EXAMPLE

The following blocks world example show how to develop an application with the frame planner. The clear-generator function is required to initialize the nonmonotonic system. The underscore symbol is used to introduce terms in Schemata in the same way as quotes does in LISP. The blocks world has three blocks A, B, C and a Table which are defined as zero arity functions. There is only one action (to put a block on another block or table) one might take, which is defined as a block function. There are two predicates On and Clear-top which are defined as properties. The initial situation is defined as Situation in which block C is on block A, block A is on table and block B is on table. The goal is to get block A on B and Block B on C.

(clear-generators)
  (define-functions (_a 0) (_b 0) (_c 0) (_Table 0) (_puton 2))
  (define-properties (_ON 2) (_Clear-top 1))
  (define-variables x y z)
  (define-situation (_ON (A C A) (ON A Table) (ON B Table)))
  (define Blocks (A B C)) (define Objects (A B C Table))
  (define goal (_ON (A B) (B C)))

The actions, their preconditions and results are defined as follows. The nil action represents no operation. The preconditions of the action (puton x y) are that the top of x and top of y must be clear. The results of the action (puton x y) is that block x is now on block y. Note that nothing is said about clearing the previous position of x. The function actions is defined as the reverse of the function results. For example, the function (actions (_On x y)) states that to achieve the goal put block x on block y in a situation K, the action (puton x y) may be done. So the goal to clear an object is defined to be achieved by moving the blocks on top of it onto the table.

  (define (precond nil) $t)
  (define (results nil) $t)
  (define (actions nil K) nil)
(define (precond (_puton x y))
  (\(\cdot\) (clear-top \(x\)) (clear-top \(y\))))

(define (results (_puton x y)) (on x y))

(define (actions (_puton x y) (puton x y))
  (define (actions (_clear-top \(O\)) \(K\))
    (if (eq? \(O\) \(Table\)) nil
      (exec-nm (\(x\)) (\(e\) (blocks))
        (\(K\) (on \(x\) \(O\))))))

To define the physical laws we say that a block cannot be on itself, no block can be placed on two objects, and two blocks cannot be put on one block.

(define physical-laws (exec-nm (\(\cdot\) (\(e\) (blocks)) (\(O\) (on \(x\))))
  (\(\forall\) \(x\) \(y\) \(z\) \(e\) \(Objects\) \(e\) \(z\) \(Objects\))
  (\(\rightarrow\) (\(\forall\) \(O\) \(Y\) \(ON\) \(Z\) \(\forall\) \(Y\) \(Z\))
    (\(\forall\) \(e\) \(blocks\) \(e\) \(blocks\) \(e\) \(blocks\)
      (\(\leftrightarrow\) (\(\forall\) \(X\) \(Z\) \(ON\) \(Z\) \(\forall\) \(X\) \(Z\)))))

Now we define the frame laws for the blocks world. The frame laws state that a block continues to hold its position of current situation (K0) if it is possible for the block to be in that position in the new situation (K) and it does not contradict the physical laws in the new situation.

(define (frame-laws K0 K) (\(\forall\) physical-laws
  (\(\forall\) \(x\) \(y\) \(z\) \(e\) \(blocks\) \(e\) \(blocks\))
    (\(\forall\) (on x y) (\(K\) (on x y)))))

The reduce-diff function must be provided by the user which is used as a closeness criterion by the actions generator to sort the actions. The following Schemata functions define that an action is preferred if the results of the action are contained in the goal.

(define (reduce-diff? r1 r2 goal) \(Sf\))
(define! (reduce-diff? r1 r2 (\(\cdot\) r1 r2)) \(Sf\))

The frame planner returns all possible plans to achieve the given goal. To get the first plan and delaying the other plans, we used the join primitive of Schemata which converts all possible plans into a stream. The Car of this stream then returns the first plan.

(define (block-example)
  (car (join (frame-planner goal situation 0 frame-laws)))))

4. BLOCKS WORLD EXAMPLE OUTPUT:

In the output, the numbers inside square brackets [xxx, yyy] represent some statistics of the Schemata system, where xxx represents the number of seconds taken and yyy represents the number of backtrack points generated during the proof of a given expression. In the output of the blocks example the partial plan is printed as "Actions", the current situation after the execution of partial plan is printed as "K." and at the end, the plan and the goal situation are returned by the blocks example. It is interesting to note that the facts are automatically added and deleted by nonmonotonic reasoning from one situation to the next as the actions are performed. This particular example shows that the frame planner do not suffer from the Sussman Anomaly, that is producing an invalid plan for some goals having parallel postconditions. In the blocks world example there are three block A, B, C and a table, the initial situation is that A and B are on the table and C is on A. The goals of getting A on B and B on C has parallel postconditions. A planner may produce a plan where B is put on C, and A is then put on B by putting B on the table, and A on B. Thus undoing the "B on C" fact of the situation. If the planner does not check the parallel postconditions, the plan so produced is invalid, since both of the goals are not true at the end of the plan.

5. THE QUANTIFICATION LOGIC Z:

The reasoning system of the frame planner uses the quantification modal logic Z that consists of the following symbols: falsity: F; truth: T; and: \(\cdot\); or: \(\lor\); for all: \(\forall\); for some: \(\exists\); not: \(\neg\); necessary: \(\Box\); synonymous: \(=\); and the following defined symbols:

\[
\begin{align*}
\alpha \rightarrow \beta &= df \left( [\alpha \rightarrow \beta] \right) \\
\alpha \iff \beta &= df \left( [\alpha \iff \beta] \right) \\
\alpha \Rightarrow \beta &= df \left( [\alpha \Rightarrow \beta] \right) \\
\alpha \Leftarrow \beta &= df \left( [\alpha \Leftarrow \beta] \right) \\
\alpha \Leftarrow \beta &= df \left( [\neg \Box \alpha \Rightarrow \beta] \right) \\
\Box \alpha &= df \left( [\alpha \Rightarrow \beta] \right) \\
\alpha \iff \beta &= df \left( [\alpha \iff \beta] \right) \\
\alpha \Leftarrow \beta &= df \left( [\neg \Box \alpha \Rightarrow \beta] \right) \\
\alpha \Rightarrow \beta &= df \left( [\alpha \Rightarrow \beta] \right) \\
\alpha \Leftarrow \beta &= df \left( [\neg \Box \alpha \Rightarrow \beta] \right) \\
\end{align*}
\]

The axioms and inference rules of Z include those of first order logic (Mendelson 1964) plus the following inference rules and axioms about necessity \(\Box\):

R0: from \(\alpha\) infer \(\Box \alpha\)
A1: \(\Box \alpha \rightarrow \alpha\)
A2: \(\Box \alpha \rightarrow \alpha\)
A3: ([(p) v (l)]=p) $\forall w (WORLD w) \rightarrow (w[p])$]
A4: ($\forall w (WORLD w) \rightarrow (w[p])$) $\rightarrow (l[p])$
A5: [WORLD]($\forall p (GEN p) \rightarrow (p=\lbrack x=\lbrack a \rbrack \rbrack)$)
for every expression $x$
A6: ($\neg (\exists x (x=\lbrack 1 \rbrack \rbrack)=\lbrack a \rbrack \rbrack)
where $x_1$ and $x_2$ are different predicates.
A7: ($\exists \exists (x=\lbrack 1 \rbrack \rbrack)=\lbrack x=\lbrack 1 \rbrack \rbrack)
where $x_1$ and $x_2$ are different functions.
A9: ($\exists \exists (x=\lbrack 1 \rbrack \rbrack)=\lbrack x=\lbrack 1 \rbrack \rbrack)

The laws R0, A1, A2 and A3 constitute an S5 modal logic which with the nonmodal laws is similar to (Carnap 1946; Bressan 1972). A4 says that a proposition is logically necessary if it is entailed by every world proposition. Laws A5, A6, and A7 axiomatize the predicates. A5 is the key axiom which says that any exhaustive conjunction of negated or unnegated distinct generators is a world if there is a sentence or expression in the formal language of $Z$ which holds when $p$ is an unnegated generator of that conjunction. It extends the A5 axiom used in (Brown 1986a) to handling quantifiers over arbitrary domains. Laws A8 and A9 axiomatize the functions. The axiom scheme A5 which has a recursively enumerable number of instances, expresses what is logically possible to the extent that it can be so expressed. What is possible with respect to a knowledgebase $K$ is expressed by the defined symbol $<K*p$ which means $K$ and $p$ together are logically possible. For example, the sentence $(\exists x \forall (\exists x \exists (x=\lbrack 1 \rbrack \rbrack)=\lbrack x=\lbrack 1 \rbrack \rbrack))$ can be derived from axiom A5 and A6 as follows. Assuming $P$ is one of the predicates and letting $a$ be $p=\lbrack (P A) \rbrack$ axiom A5 of $Z$ becomes: (WORLD ($\forall p (\exists x \exists (x=\lbrack 1 \rbrack \rbrack)=\lbrack x=\lbrack 1 \rbrack \rbrack)))$ where $\exists$ is the rest of the GEN definition. This gives: (WORLD ($\forall p (\exists x \exists (x=\lbrack 1 \rbrack \rbrack)=\lbrack x=\lbrack 1 \rbrack \rbrack)))$ where $\exists$ is the rest of the GEN definition. Thus the reflexive equivalence: $K=a$ where $K$ may occur within $a$, axiomatizes the knowledgebase $K$. Any equation of the form $K=\beta$ where $K$ does not occur in $\beta$ and which implies $K=\alpha$, is a solution to the original equivalence.

6. CONCLUSION

The frame planner extends the classical planning paradigms by nonmonotonic reasoning about actions using the quantification modal logic $Z$ and the Deduction Language Schemata. It uses frame axioms to carry the facts from the current situation to the next situation, which makes the action description simpler. The Frame planner, as the name suggests, handles the frame problem in a more logical way. The frame planner can produce a stream of all possible plans to solve a given goal. It does not suffer from the Sussman anomaly, that is producing an invalid plan for some goals having parallel postconditions.

7. ACKNOWLEDGEMENTS

We thank Dr. Sung Park and Dr. James Miller for their help in this research.

8. REFERENCES

HOW FUZZY SHOULD A NEURON BE?

Joseph M. Barone
Loki Software Inc.
P.O. Box 71
Liberty Corner, N.J. 07938

Abstract: For the most part, fuzzified models of real-world phenomena have not been completely integrated with or carefully related to the distinguishing features of their hard counterparts. This paper explores some of the consequences of this approach as they emerge in fuzzified treatments of neural network models (esp. perceptrons). It is shown that syntactic fuzziness which is based more directly on the semantic properties of the data under consideration has the potential to yield a model with fuller explanatory power.

Keywords: Fuzzy sets, neural networks, rough sets, perceptron.

The question of how best to fuzzify a model characterized by non-fuzzy mathematics (assuming there is reason to fuzzify it in the first place) has usually been addressed from a practical point of view. What is done, typically, is to isolate the points at which the model is open to set-theoretical treatment and then to fuzzify the characterization of those points in some appropriate fashion. This amounts to the introduction of a fuzzy syntax for a model whose semantics is left entirely hard (or entirely open - "semantics" is used here to refer to the formal specification of the kinds of elements to which the syntax (and hence the model) may be meaningfully applied). Thus, for instance, Kacprzyk and Orlovski (1987) fuzzify various optimization models by treating the goals and constraints to be optimized (as well as the objective function) as fuzzy sets and then performing the optimization via fuzzy operations (esp. min); it is simply assumed that the "set of relevant alternatives" (ibid. p. 53) is amenable to the analysis described by virtue of the fact that fuzzy (constraint and goal) membership functions can be generated for them (this is in effect the sum-total of the semantic component of the fuzzy model). The validity of the entire approach rests on the provability of the claim that models in which the set-theoretical components and the essential operations are fuzzified yield better results than their hard counterparts (in practical situations comparison of results may be straightforward, but for the general case Bezdek's caveat regarding validity functionals over (the fuzzy partition) U (mutatis mutandis) is worth keeping in mind (Bezdek 1981, p. 137)).

It should be noted that this question is distinct from the question of how best to reveal and describe the (fuzzy) clusters into which a set of related data vectors may most reasonably be partitioned. Fuzzy clustering, in the context just adumbrated, is a semantic (actually pragmatic) endeavor whose most effective methods may well be related to the techniques of factor analysis (this connection is carefully developed in Bezdek 1981, esp. Section 23). This, of course, is not to minimize the importance of the pragmatic component of fuzzy set theory, since it may be the key to deciding whether there is any reason to fuzzify a model in the first place (if the data is fuzzy then a fuzzy model is probably called for).

This question is also distinct from (though possibly related to) the question of whether or not the fuzzy character of the data set under investigation may dictate the choice of (hard) model or
facilitate its operation. Thus, for instance, Negoiu (1979) shows that the lattice structure of a fuzzy "carrier" may greatly facilitate (via "pullback") linear programming operations on the data covered by the carrier. The point he makes, though, is a semantic one since his analysis proceeds (connects to the syntactic properties of the linear programming model) from the properties of lattices rather than from the properties of fuzzy sets.

Perhaps the most explicit (and interesting) attempts to characterize a coherent fuzzy model have been made in the area of fuzzy control (Sugeno and Takagi 1983). More relevant to our purpose, though, are the clear connections which have been established between the syntactic and semantic character of certain fuzzy "concepts" in the general sense (apart from any particular model). Several examples can be found in Kim and Roosh (1987), for instance, among them the syntactic realization of the semantic concept of the order of fuzzy elements as a permutation of an idempotent power of the matrix of inter-element dominance connections (p. 125).

A particularly good example is developed at length in Kandel (1986, pp. 180 ff.), and is worth examining in some detail in the present context. Let $x_1, x_2, \ldots, x_n$ be a set of data points and associate with each pair of points a fuzzy number which represents the subjective similarity which obtains between them; clearly, this arrangement can be represented by a matrix, e.g. (M):

<table>
<thead>
<tr>
<th></th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>1</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>$x_2$</td>
<td>0.5</td>
<td>1</td>
<td>0.9</td>
<td>0.7</td>
</tr>
<tr>
<td>$x_3$</td>
<td>0.5</td>
<td>0.9</td>
<td>1</td>
<td>0.7</td>
</tr>
<tr>
<td>$x_4$</td>
<td>0.5</td>
<td>0.7</td>
<td>0.7</td>
<td>1</td>
</tr>
</tbody>
</table>

which is idempotent. The connection between the concept "more closely related" then, and the (surface) syntactical arrangement (matrix) M is provided by the "raise to an idempotent power" transformation. (e.g., the fact that $x_3$ is more closely related to $x_4$ than $x_1$ is to $x_3$ is apparent in $M^2$ but not in M). It should be noted that Kandel's methodology is strictly fuzzy, and that the connections established depend on this fact.

It is possible to think of the semantic component of a neural network as a (rough set) system (Waiselwaska 1988) in which the objects are a set of row vectors, the attributes are (marked by) the elements in the row, the values range over (say) the positive real numbers, and the information function is given by an assignment of values to the elements, e.g. (S):

<table>
<thead>
<tr>
<th></th>
<th>$a_1$</th>
<th>$a_2$</th>
<th>$a_3$</th>
<th>$a_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$o_1$</td>
<td>1.2</td>
<td>0.4</td>
<td>2.5</td>
<td>4.3</td>
</tr>
<tr>
<td>$o_2$</td>
<td>0.9</td>
<td>1.1</td>
<td>0.8</td>
<td>4.5</td>
</tr>
<tr>
<td>$o_3$</td>
<td>0.7</td>
<td>0.4</td>
<td>4.8</td>
<td>2.2</td>
</tr>
<tr>
<td>$o_4$</td>
<td>0.9</td>
<td>1.4</td>
<td>1.4</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Following Waiselwaska's lead, let the vector partition provided by the expert be described by a new attribute applied to each of the objects in the set, e.g. (S'):

<table>
<thead>
<tr>
<th></th>
<th>$a_1$</th>
<th>$a_2$</th>
<th>$a_3$</th>
<th>$a_4$</th>
<th>$a_p$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$o_1$</td>
<td>1.2</td>
<td>0.4</td>
<td>2.5</td>
<td>4.3</td>
<td>1</td>
</tr>
<tr>
<td>$o_2$</td>
<td>0.9</td>
<td>1.1</td>
<td>0.8</td>
<td>4.5</td>
<td>0</td>
</tr>
<tr>
<td>$o_3$</td>
<td>0.7</td>
<td>0.4</td>
<td>4.8</td>
<td>2.2</td>
<td>0</td>
</tr>
<tr>
<td>$o_4$</td>
<td>0.9</td>
<td>1.4</td>
<td>1.4</td>
<td>1.0</td>
<td>1</td>
</tr>
</tbody>
</table>

In this context, a perceptron may be thought of as a decision algorithm which relates (syntactically) the system without the expert-supplied attribute to the system with the expert-supplied attribute. Of course, actually writing a set of decision rules which is equivalent to a perceptron may be difficult, but this framework renders the process of fuzzification fairly clear. Fuzzification may be described as
the addition of another attribute to each of the objects in the set under consideration, this attribute being the degree of membership of each of the objects in a fuzzy set assigned by some method external to the system itself, e.g. \((S'')\):

<table>
<thead>
<tr>
<th></th>
<th>a1</th>
<th>a2</th>
<th>a3</th>
<th>a4</th>
<th>ap</th>
<th>a5</th>
</tr>
</thead>
<tbody>
<tr>
<td>o1</td>
<td>1.2</td>
<td>0.4</td>
<td>2.5</td>
<td>4.3</td>
<td>1.0</td>
<td>0.44</td>
</tr>
<tr>
<td>o2</td>
<td>0.9</td>
<td>1.1</td>
<td>0.8</td>
<td>4.5</td>
<td>0</td>
<td>0.88</td>
</tr>
<tr>
<td>o3</td>
<td>0.7</td>
<td>0.4</td>
<td>4.8</td>
<td>2.2</td>
<td>0</td>
<td>0.51</td>
</tr>
<tr>
<td>o4</td>
<td>0.9</td>
<td>1.4</td>
<td>1.4</td>
<td>1.0</td>
<td>1.0</td>
<td>0.64</td>
</tr>
</tbody>
</table>

A fuzzy perceptron, then, is nothing more than a decision algorithm which relates \(S'\) (non-fuzzy) to \(S''\) (fuzzy). In the most general sense, and in this context, the way to fuzzify a neuron is to "blur" (see Dubois and Prade 1979, p. 36) the neuron's syntactic and semantic structure to accommodate fuzzy pragmatics.

A number of recent papers have proposed schemes whereby fuzzy logic and neural network logic work together to achieve some goal (typically pattern recognition or control). These proposals have differed considerably in the degree to which the fuzzy component and the neural component interact to achieve the desired result. In Gupta, Pedrycz, and Kiszka (1989), for instance, fuzzy logic is used to set up the input to a (non-fuzzy) neural network, and the output of the network is taken to have fuzzy significance; the operation of the neural net itself, however, is entirely traditional. The neural network, in other words, is used only as a tool to perform fuzzy computation. At the other extreme, Yamakawa and Tomoda (1989) have proposed a totally fuzzy neuron in which both training and operation are fully characterized by fuzzy operations. Other researchers have taken intermediate positions, that is, positions where the operation of the neural network is partially fuzzified. Kuncicky and Kandel (1989), for example, describe a model with a fuzzy activation function, while Keller and Hunt (1985) introduce a fuzzy correction step into the neural network's training phase (a more refined and elaborate version of their model may be found in Keller and Qiu 1988). Taking still another approach, Huntsberger and Ajimaransee (to appear) demonstrate that continuous-valued output can be effectively generated by self-organizing neural nets using a fuzzy output function and fuzzy feedback. It is not the purpose of this paper to support one or another of these alternative approaches, but simply to demonstrate that introduction of fuzzy concepts into the neural network domain must be done with care lest the inherent functionality of the neural net be compromised by its fuzzification.

Keller and Hunt (1985) describe a method for introducing fuzzy values into learning in the perceptron. Their method consists, essentially, of using the fuzzy strength of membership in a cluster (as measured by distance from the cluster mean) as a factor in the perceptron learning rule. Using their method, training vectors which belong strongly to either cluster influence learning much more than vectors which belong only weakly to both clusters, and vectors which exactly straddle the fence (therefore) have no influence on learning whatsoever. The success of Keller and Hunt's method depends strongly on the preexistence of a (reasonable) hard partition. The fuzzy partition, in other words, and the corresponding fuzzy membership values assist only in locating a separating hyperplane (determining the value of the weight vector) if such a hyperplane exists; if it does not exist, the fuzzy membership values assist in arriving at the "best" possible approximation to a decision boundary. The fuzzy values are not used in any way to assign training vectors to the (hard) classes in the first place. Keller and Hunt, therefore, posit a strict separation between hard and fuzzy roles in the operation of their fuzzy perceptron; the training vectors are separated by (some) hard criterion and the fuzzy partition is used strictly as a training catalyst.

The validity of this position was examined by applying Keller and Hunt's methods to training data sets which are basically (hard and fuzzy) partitionable but which contain at least one vector with equal membership in both classes (i.e., whose fuzzy membership vector is 0.50, 0.50). This being the case, one should expect perceptron learning to be relatively unaffected by the (hard) assignment of these ambiguous vectors to one or the other of the (hard) classes. In other words, it should be possible to achieve essentially equivalent separation of classes regardless of where (into which hard class) the ambiguous vector(s) are placed, since they are close to the line of
separation to begin with.

To take a specific case, the "butterfly" data (Figure 1) analyzed extensively in Bezdek (1981) was examined using the fuzzy membership values generated by Bezdek's Algorithm 9.1 in Keller and Hunt's version of the perceptron learning rule (see esp. pp. 50-53). The butterfly data consists of fifteen points in R2 (see Table 1); by almost any clustering criterion, these points fall naturally into two classes of seven points each. The remaining point (vector #8 in Table 1) tends to straddle the class boundary more or less exactly depending on the fuzzy clustering method used and its associated weighting and normalizing parameters. For purposes of discussion, an algorithm was chosen (Bezdek's 9.1) which places the vector squarely on the (2-) class boundary; from a fuzzy point-of-view, this is almost certainly the most reasonable description of the data (it should be emphasized that this is not the partitioning method used by Keller and Hunt).

Keller and Hunt's perceptron learning algorithm was run using the butterfly data just described under four different circumstances:

1. "fuzzy influenced" - vector #8 in group 2 (run ef2)

2. "hard only" - vector #8 in group 2 (run eh2)

3. "fuzzy influenced" - vector #8 in group 1 (run ef1)

4. "hard only" - vector #8 in group 1 (run eh1)

Note: Group 2 vectors are those which have been multiplied by -1 in accordance with Keller and Hunt's method and all the training vectors are "augmented" with a 1 (again following Keller and Hunt). All runs reached equilibrium (converged) quickly, except for run ef1 which did not converge at all.

If the fuzzy and hard partitions operate independently relative to the perceptron learning algorithm, one should expect to find little difference among the four cases beyond slight variations caused by the stepwise rate adjustments in the weight vector during the learning process (the fuzzy-influenced learning process adjusts in smaller increments, so when it is training on an easily-located hyperplane the hard variety gets there a little faster).

This expectation, however, turns out not to be the case. Both "hard only" runs work well, but the "fuzzy influenced" case in which vector #8 is assigned to group 1 does poorly and routinely misclassifies this vector (see Table 1). This, of course, is hardly surprising. Since the ambiguous vector can have no influence on the learning process, the perceptron cannot arbitrarily "learn" what to do with it. The fuzzy component of the perceptron, in other words, dictates (in effect) the exact placement of the hyperplane and hence influences the acceptability of possible hard partitions; it has instantiated itself into the perceptron learning process in general.

It would appear that this situation can be dealt with effectively by using a different fuzzy partition to drive the perceptron learning process. This partition is calculated using the assumption that the hard centroid vectors induced by the (assigned) hard partition are the centroids around which the training vectors must cluster. The method used is the same as the one used to calculate Gunderson's separation coefficient (see Bezdek 1981, pp. 137 ff. and esp. p. 140) except that the hard partition U' is not calculated but rather given by the (arbitrary) assignment of the training vectors to two hard classes. The resultant fuzzy partition (U'' in Bezdek's notation) is (in some sense) more closely allied to the hard partition than the original (independent) fuzzy partition was, though, of course, it is also less "correct" from a purely fuzzy point of view. These "altered" partitions drive the perceptron directly toward the separating hyperplane (if it exists, of course) regardless of the hard partition selected (since the ambiguous vector is now assigned (weakly but distinctly) to the appropriate fuzzy cluster).

The new fuzzy partition for the case in which vector #8 is assigned to group 1 is shown in Table 2, while the fuzzy partition for the case in which vector #8 is assigned to group 2 is shown in Table 3. When these fuzzy partitions are used, the perceptron quickly converges to an appropriate weight vector in each of the "fuzzy influenced" cases. Of course,
since the non-fuzzy perceptron converges easily in both cases, one might wonder why the perceptron should be fuzzified in the first place. It would appear, at the very least, that fuzzification of the partition of the training vector set may result in more rapid convergence of the perceptron process (see Keller and Hunt 1985) and may provide a measure of the feasibility of the search for a separating hyperplane that the perceptron must perform.

In this connection, it is convenient to think of the "one-step" fuzzification adumbrated above as the first fuzzy moment of area of the Voronoi polygons associated with the data points (the vectors) under investigation (see Tuceryan and Jain (1990)). From this point of view, the entities being fuzzified are the edges of the (hard) polygons which are (in effect) qualified and transcended (and hence fuzzified) by the degree of proximity of the area they enclose to the two hard (expert-induced) centroids associated with the data set as a whole. This fuzzy tessellation in turn provides a framework within which the syntactic realization (effect) of the semantic fuzzy partitioning can be actualized. As such, of course, it can be made to influence the (syntactic) operation of the perceptron relaxation (learning) process. The influence of the fuzzy tessellation on the operation of the perceptron can be considered as equivalent to the influence of a "hard" tessellation on (e.g.) texture analysis.

One or two simple examples serve to illustrate the usefulness of this approach. For the "straight line" data set shown in Figure 2, the hard perceptron requires twelve iterations to find the separating hyperplane, while the perceptron influenced by the first fuzzy moment finds it after just one. For "touching rectangles" data, the first fuzzy moment for a reasonable ("expert"-supplied) clustering (Figure 3) is shown in Table 4 and the first fuzzy moment for an unreasonable ("expert"-supplied) clustering (Figure 4) is shown in Table 5. The relative fuzziness of the two partitions provides a clear indication of the feasibility of finding a separating hyperplane for the partition.

It appears, therefore, that in the case of Keller and Hunt's version of the fuzzy perceptron, at least, some hard partitions of the training vector sets, though otherwise reasonable, may be incompatible with fuzzified training of the "independent" type. It may be the case, then, that fuzzified perceptrons may be able to learn only a subset of the set of linear separations possible over a set of (training) vectors and learnable by a "hard" perceptron unless the fuzzification is allied more intimately to the desired result. The general significance of this observation remains to be established, but it should at least be taken into consideration by researchers working on the fuzzification of neural networks. In any case, careful introduction of fuzzy constructs into the semantic component of a neural net should allow principled integration of fuzziness into existing hard models such as the one promulgated by Baldi and Hornik (1989).

References


| TABLE 1: BUTTERFLY DATA, FUZZY PARTITION, and PERCEPTRON CLASSIFICATION ERRORS |
|---|---|---|---|---|---|---|---|---|---|---|
| v1 | v2 | grp | f1 | f2 | ef2 | ch2 | ef1 | ch1 |
| vec1 | 0.0 | 0.0 | 1 | .99 | .01 | 1 | 1 | 1 | 1 |
| vec2 | 0.0 | 2.0 | 1 | .98 | .02 | 0 | 1 | 0 | 1 |
| vec3 | 0.0 | 4.0 | 1 | .99 | .01 | 2 | 1 | 2 | 0 |
| vec4 | 1.0 | 1.0 | 1 | .86 | .14 | 2 | 1 | 2 | 4 |
| vec5 | 1.0 | 2.0 | 1 | .85 | .15 | 0 | 0 | 0 | 0 |
| vec6 | 1.0 | 3.0 | 1 | .86 | .14 | 0 | 0 | 0 | 0 |
| vec7 | 2.0 | 2.0 | 1 | .67 | .33 | 3 | 0 | 3 | 5 |
| vec8 | 3.0 | 2.0 | 1 | .50 | .50 | 2 | 2 | * | 3 |
| vec9 | 4.0 | 2.0 | 2 | .33 | .67 | 1 | 0 | 1 | 6 |
| vec10 | 5.0 | 1.0 | 2 | .14 | .86 | 1 | 0 | 1 | 0 |
| vec11 | 5.0 | 2.0 | 2 | .15 | .85 | 0 | 0 | 0 | 0 |
| vec12 | 5.0 | 3.0 | 2 | .14 | .86 | 0 | 0 | 0 | 0 |
| vec13 | 6.0 | 0.0 | 2 | .01 | .99 | 0 | 0 | 0 | 0 |
| vec14 | 6.0 | 2.0 | 2 | .02 | .98 | 0 | 0 | 0 | 0 |
| vec15 | 6.0 | 4.0 | 2 | .01 | .99 | 1 | 0 | 1 | 1 |
| conv | - | - | - | - | - | yes | yes | no | yes |

* This vector was routinely misclassified during this training exercise.

* Vector 8 was assigned to grp 2 for trials ef2 and ch2 and to grp 1 for trials ef1 and ch1.
### TABLE 2: FUZZY PARTITION: BUTTERFLY DATA (VECTOR 8 IN GROUP 1)

<table>
<thead>
<tr>
<th>v1</th>
<th>v2</th>
<th>v3</th>
<th>v4</th>
<th>v5</th>
<th>v6</th>
<th>v7</th>
<th>v8</th>
<th>v9</th>
<th>v10</th>
<th>v11</th>
<th>v12</th>
<th>v13</th>
<th>v14</th>
<th>v15</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.86</td>
<td>0.97</td>
<td>0.86</td>
<td>0.95</td>
<td>1.00</td>
<td>0.95</td>
<td>0.92</td>
<td>0.57</td>
<td>0.16</td>
<td>0.06</td>
<td>0.01</td>
<td>0.06</td>
<td>0.13</td>
<td>0.02</td>
<td>0.13</td>
</tr>
<tr>
<td>0.14</td>
<td>0.03</td>
<td>0.14</td>
<td>0.05</td>
<td>0.00</td>
<td>0.05</td>
<td>0.08</td>
<td>0.43</td>
<td>0.84</td>
<td>0.94</td>
<td>0.94</td>
<td>0.87</td>
<td>0.98</td>
<td>0.87</td>
<td></td>
</tr>
</tbody>
</table>

### TABLE 3: FUZZY PARTITION: BUTTERFLY DATA (VECTOR 8 IN GROUP 2)

<table>
<thead>
<tr>
<th>v1</th>
<th>v2</th>
<th>v3</th>
<th>v4</th>
<th>v5</th>
<th>v6</th>
<th>v7</th>
<th>v8</th>
<th>v9</th>
<th>v10</th>
<th>v11</th>
<th>v12</th>
<th>v13</th>
<th>v14</th>
<th>v15</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.87</td>
<td>0.98</td>
<td>0.87</td>
<td>0.94</td>
<td>0.99</td>
<td>0.94</td>
<td>0.84</td>
<td>0.43</td>
<td>0.08</td>
<td>0.05</td>
<td>0.00</td>
<td>0.05</td>
<td>0.14</td>
<td>0.03</td>
<td>0.14</td>
</tr>
<tr>
<td>0.13</td>
<td>0.02</td>
<td>0.13</td>
<td>0.06</td>
<td>0.01</td>
<td>0.06</td>
<td>0.16</td>
<td>0.57</td>
<td>0.92</td>
<td>0.95</td>
<td>1.00</td>
<td>0.95</td>
<td>0.86</td>
<td>0.97</td>
<td>0.86</td>
</tr>
</tbody>
</table>

### TABLE 4: FUZZY PARTITION: TOUCHING RECTANGLES (REASONABLE CLUSTER)

<table>
<thead>
<tr>
<th>v1</th>
<th>v2</th>
<th>v3</th>
<th>v4</th>
<th>v5</th>
<th>v6</th>
<th>v7</th>
<th>v8</th>
<th>v9</th>
<th>v10</th>
<th>v11</th>
<th>v12</th>
<th>v13</th>
<th>v14</th>
<th>v15</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.83</td>
<td>0.95</td>
<td>0.92</td>
<td>0.79</td>
<td>0.95</td>
<td>0.85</td>
<td>0.79</td>
<td>0.43</td>
<td>0.14</td>
<td>0.15</td>
<td>0.10</td>
<td>0.14</td>
<td>0.09</td>
<td>0.06</td>
<td>0.15</td>
</tr>
<tr>
<td>0.17</td>
<td>0.05</td>
<td>0.08</td>
<td>0.21</td>
<td>0.05</td>
<td>0.17</td>
<td>0.21</td>
<td>0.57</td>
<td>0.86</td>
<td>0.85</td>
<td>0.94</td>
<td>0.86</td>
<td>0.91</td>
<td>0.94</td>
<td>0.85</td>
</tr>
</tbody>
</table>

### TABLE 5: FUZZY PARTITION: TOUCHING RECTANGLES (UNREASONABLE CLUSTER)

<table>
<thead>
<tr>
<th>v1</th>
<th>v2</th>
<th>v3</th>
<th>v4</th>
<th>v5</th>
<th>v6</th>
<th>v7</th>
<th>v8</th>
<th>v9</th>
<th>v10</th>
<th>v11</th>
<th>v12</th>
<th>v13</th>
<th>v14</th>
<th>v15</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.50</td>
<td>0.50</td>
<td>0.50</td>
<td>0.50</td>
<td>0.50</td>
<td>0.50</td>
<td>0.50</td>
<td>0.50</td>
<td>0.00</td>
<td>0.50</td>
<td>0.50</td>
<td>0.50</td>
<td>0.50</td>
<td>0.50</td>
<td></td>
</tr>
<tr>
<td>0.50</td>
<td>0.50</td>
<td>0.50</td>
<td>0.50</td>
<td>0.50</td>
<td>0.50</td>
<td>0.50</td>
<td>1.00</td>
<td>0.50</td>
<td>0.50</td>
<td>0.50</td>
<td>0.50</td>
<td>0.50</td>
<td>0.50</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 1:** Butterfly data

**Figure 2:** Straight line

**Figure 3:** Touching rectangles reasonable clustering

**Figure 4:** Touching rectangles unreasonable clustering

183
Constraint Management for Integrated Systems

Elenita Elion Silverstein and Peggy Sun
Mc Donnell Douglas Corporation
Artificial Intelligence Group
10855 Business Center Dr., Cypress, CA 90630 USA
(714) 229-3144

KEYWORDS: knowledge representation, knowledge-based systems, system integration, search, heuristics, constraints, constraint satisfaction

ABSTRACT

There is an increasing number of knowledge-based systems being used to solve problems in the domains of design, planning, and scheduling. From the manufacturing point of view, these domains can be considered as phases of a product life cycle. Therefore, it is logical that these different knowledge-based applications should be integrated into one system. A constraint management system can help in the task of integration by providing a standard way of representing and operating on constraints that impact the problem-solving processes of several knowledge-based applications. This paper describes a constraint management system that we have developed and how it has been used to create a knowledge-based application for circuit layout design.

INTRODUCTION

Artificial Intelligence has traditionally been concerned with complex problem domains that involve many different types of interacting constraints. These include the domains of design, planning, and scheduling. The usual approach to these types of problems is to create knowledge-based systems that take advantage of domain-specific heuristics and constraints to converge on a solution quickly. These systems generally concentrate on a specific domain, and build in only the constraints that are needed to solve the particular problem at hand. As a result, customized constraint models are created for each type of application.

We are concerned with the integration of the applications of these domains. From a manufacturing point of view, planning, scheduling, and design are phases in a product life cycle that interact and need to pass information among themselves. Solutions produced have to be globally optimal, and should consider the interdependencies among the different phases of the product life cycle to minimize retractions of decisions made earlier in the process.

Before a system can be able to integrate different domain applications, it must have the ability to uniformly represent and operate on concepts shared by multiple applications. A constraint management system can fulfill this requirement because it communicates with all the phases of the product life cycle and can evaluate and propagate constraints within and among the appropriate processes.

A constraint management system that globally operates on constraints of several applications cannot address the disadvantages of the customized approach to managing constraints. These disadvantages stems from the fact that the customized approach tailors constraints to the particular problem at hand, resulting in incompatible representations of the different objects in the problem domains. The applications of these domains cannot easily talk to one another because of their inconsistent representations, thus the tendency to produce solutions that are locally consistent but not globally optimal. Furthermore, interactions between processes cannot be handled gracefully without rerunning affected applications. This problem is further compounded because these applications operate in a dynamic environment where decisions are retracted and asserted at different places, and new solutions must be produced as a result of changing data. For example, changes in the design specifications of a part (i.e., in the design phase) may impact the type of machines that have to be scheduled in order to produce the part (i.e. in the scheduling phase).

Proceedings of the 3rd Florida Artificial Intelligence Research Symposium, 1990, by the Florida AI Research Society

c. 1990 FLAIRS 0-9620-1732-9/90/0350-0184

184
In this paper, we present a system called CONSYST. It is our initial effort at integrating systems by providing a standard constraint representation and a set of operations. CONSYST is a constraint management system that has the following functionalities:

- provides a constraint representation template for creating and editing constraints
- classifies constraints based on their function in the problem domain
- evaluates constraints automatically
- performs error handling and constraint relaxations or provides advice whenever a constraint is violated.
- provides different constraint-handling functions depending on the problem-solving models used by the problem solver of the application

Representation of Constraints

Although constraints are classified into different types, as described below, all constraints share one basic representation scheme. A constraint is represented as a frame with the attributes in Table 1.

Each constraint is associated with one or more objects (specified in the constraint.of slot). Objects can be artifacts of the problem domain such as Boolean gates or pins (in the circuit layout domain to be described later). Objects can also be other constraints. In this manner, constraints can be activated upon the evaluation of another constraint.

If the constraint is associated with an artifact, it is typically activated when one or more attributes of that artifact changes. For instance, a constraint on the location of a Boolean gate in a circuit board would be activated whenever the x or y coordinates of that gate is altered. This relationship is explicitly stated in the constraint.parameter slot, which would list the objects and corresponding parameters that affect the constraint.

A Lisp function that evaluates the constraint is associated through the tester or the relaxation slot, depending upon the type of constraint that is involved. This Lisp function is responsible for detecting different error situations and generating advice (Mittal 1986) in the form of alternative parameter values. The output of the Lisp function is an ordered set of advice, and their corresponding utility values which are a relative numerical preference for the associated advice, or nil, if the constraint is not violated.

When a constraint is violated, an instance of the constraint is created and the advice generated is stored in the advice slot of that instance. This is accessed by the constraint manager and entries are tried, in sequence, for as long as the constraint is violated. If a piece of advice has been tried and still produces an error, the constraint manager first checks to see if there is an instance of that constraint that has already been created and then checks to see if there is still some advice that has not been tried.

A knowledge-based application that is organized around abstraction levels can have constraints that are activated by the current abstraction level at which the problem solver is operating. To associate a constraint with a particular problem-solving phase or abstraction level, the duration slot is used to store a Lisp function which checks if the constraint is activatable.

The other attributes listed in Table 1.0 allow more complex operations on the constraints, such as the ability to automatically generate more relaxed versions of the constraint evaluation function (relax.fcn), to create constraint instances used as part of a constraint satisfaction network (generator), and to
evaluate full or partial solutions produced by the problem-solver (advice.utility.fcn, relax.utility.fcn, etc.). Evaluating a solution requires an evaluation function that takes into consideration the utility values of all the advice taken, the relaxation functions used to evaluate a constraint, and whether or not certain types of constraints (such as preferential constraints) have been satisfied.

Bookkeeping is done to keep track of the errors generated and the advice used by the problem solver. This information is used to detect dead-end situations. If the problem solver has generated too many constraint violating solutions and the amount of advice given exceeds a threshold, then the process is halted since the system would not find a solution within a reasonable amount of time. At this point, CONSYST looks for constraints that can be relaxed, and presents these to the user. The user can either provide a new constraint value or let CONSYST compute them using the relaxation function of that constraint.

Classification of Constraints

Constraints are categorized into a hierarchy of types based on their function with respect to the problem solver (Fox 1987) (see Figure 1). Constraints in CONSYST are either relaxable or nonrelaxable. A relaxable constraint can be altered in some prescribed manner if it is violated by a solution. The relaxable components of a constraint are represented by relaxable variables whose values are determined by the relax.fcn or the user of the application. In the application that we have developed, a dialog box appears on the screen which asks for alternative values of the relaxable variables of a violated constraint.

Nonrelaxable constraints cannot be changed in the same way that relaxable constraints are loosened to accommodate a solution. There are two kinds of nonrelaxable constraints. Required constraints are hard restrictions on a solution. Values that violate this type of constraint have to be retracted. Preferential constraints are constraints that do not have to be satisfied but guide the problem solver towards good and optimal solutions.

Finally, heuristics are modelled as solution generative constraints that assert facts in the knowledge base of the problem domain.

CONSYST and the Circuit Layout Generator Prototype

To demonstrate the general utility of CONSYST, we are planning to prototype several application domains for the different phases of a product life cycle (design, capacity planning, process planning, production scheduling, etc.). We have chosen the area of design for our first prototype. A circuit layout program has been developed that works with CONSYST to design the layout of a circuit.

Circuit Layout Domain The application we have created is a problem-solving system for generating aesthetic circuit diagrams. The input to the system consists of the logical connections that make up a circuit (see Figure 2).

The primitive components of the circuit are restricted to Boolean gates (AND, OR, NAND, NOR, XOR, INVERTER, multiplexers) collectively called parts. The output is a physical layout of the circuit components. Gates are located on a grid, and wires routed between them (see Figure 3).

Circuit Layout Constraints Several constraints are imposed on the problem-solving process for circuit layout design. Most of these constraints concern the spatial locations of gates and wires. For example, there are constraints that prohibit the overlapping of parts and wires. Gates are constrained to certain sizes, wires and parts cannot be too near the border of the circuit board, etc.
In addition, the problem solver makes use of several heuristics for the different abstraction levels of the circuit-layout problem. These constraints provide the problem solver with partial solutions that can be used to generate a more detailed solution. Heuristic constraints are used to evaluate the current state of a solution. Examples of these are aesthetic constraints that specify preferences for laying out the parts of a circuit. For example, wires connecting parts should have as few bends as possible, parts should be evenly spaced throughout the circuit box, and so on.

**System Architecture** CONSYST is a separate module that interfaces with the problem solver of an application (Fig. 4). It can respond immediately to any changes that occur in the problem state through the attachment of daemons to constrained objects. Because of the daemon approach, CONSYST is nearly transparent to the application. However, communication must exist between the problem solver and CONSYST so that some control can be imposed on the solution generation. The problem solver acts as a strawman that generates candidate solutions which are pruned by CONSYST. Thus, the problem solver does not do any error checking or validation of intermediate solutions. Our motivation in doing this is to unload much of the responsibility of handling the constraint management control issues from the problem solver onto the constraint manager. However, CONSYST is flexible enough a system to allow the developer of an application to build arbitrarily complex constraints into the problem solver without interfering with or being interfered by the actions of CONSYST.

**Figure 4 System Architecture**

This work is similar to research being conducted by Serrano, et al, at MIT (Sriram 1989), Fox (Fox 1989) and Guesgen (Guesgen 1989). What distinguishes CONSYST from these other systems is our emphasis on system integration as a long term goal. The CONMAN system is being developed by MIT as part of DESIGN-KIT, which is envisioned as a computer-aided engineering environment primarily used for design applications.

We have also been influenced by Fox's earlier work on constraint-directed search, specifically the constraint representation developed for the ISIS and OPIS systems (Fox 1987). We have added features to Fox's representation that provide additional problem-solving power, such as the ability to provide context-sensitive advice to the problem solver whenever a constraint-violating situation can be analyzed and repaired. Fox is concerned with creating and formalizing a new problem-solving model called Constrained Heuristic Search (CHS) that combines constraint satisfaction with heuristic search in order to reduce search complexity. We provide a similar capability of performing heuristic search using constraints, but consider it as only one of the problem-solving techniques to be handled by CONSYST. This is driven by our eventual goal of system integration and the fact that different systems use different models of problem solving.

Guesgen's work on CONSAT is about developing a language for expressing constraint satisfaction problems (CSP). The system uses several problem solving control strategies to efficiently compute values of the undetermined variables by exploiting
the current CSP optimization techniques such as arc consistency. We are planning on including a capability for expressing problems as constraint satisfaction networks.

FUTURE RESEARCH

One of our objectives is to produce a useful tool for developers to model constraints needed in various applications. Thus, CONSYST must be able to support a range of problem-solving techniques that use constraints. Some of these techniques deal with Constraint Satisfaction Problems (CSP), heuristic search, partial constraint networks, and approximations. Different submodules corresponding to these techniques will be built into CONSYST and can be invoked by an application. And in addition, a more sophisticated truth maintenance system needs to be built in to support these modules.

Future efforts will also be directed towards managing multiple systems by passing constraint information to and from different applications. We will demonstrate these functionalities by developing a scheduling prototype using CONSYST that can communicate information to another application. The current version of CONSYST only allows it to communicate with one application at a time. The next logical step is to allow coordination with several applications.

CONCLUSION

One of the main concerns in the manufacturing sector is to be able to create integrated systems. This requires the ability to communicate information among the applications of the different phases that span a product life cycle. Information that must be shared includes the constraints that interact with and affect the solutions produced by each phase.

To this end, we have developed a constraint management system called CONSYST that is used in conjunction with a problem solver of an application. It provides a constraint representation scheme and a mechanism that evaluates and propagates constraints, and handles errors produced by the problem solver. Various classes of constraints can be represented in CONSYST, such as relaxable and non-relaxable constraints.

We view CONSYST as a developer's toolbox that contains several submodules which support various types of problem-solving activity using constraints. The current implementation supports a form of constrained heuristic search and was used to develop a circuit layout prototype.

REFERENCES:


A MODEL BASED POWER CONTROLLER
FOR THE MANNED SPACE STATION

Robert A. Morris, Shrikant C. Raval
Dept. of Computer Science
The Florida Institute of Technology
Melbourne, Fl., 32901

Avelino J. Gonzalez
Dept. of Computer Engineering
University of Central Florida
Orlando, Fl. 32816

Keywords: Architectures for Knowledge-Based Systems,
Implementation Issues

ABSTRACT
This paper describes an Intelligent Critical Power Controller (ICPC) for managing various critical aspects of onboard power generating systems of manned space stations, aspects such as knowledge-based load scheduling, fault identification and isolation, and maintenance of power to life support systems and experimental loads. The ICPC is intended to forecast future power needs, forecast load characteristics based on past knowledge, forecast and recover from faults and their consequences, and enable onboard management of a power system without engaging the crew, thereby freeing them for other, less mundane tasks. This paper focuses on the structure of an equipment knowledge base for an ICPC, which comprises the model of the space station devices and their behavior.

1. INTRODUCTION
A manned space station will require an increased contribution by automated systems in order to decrease the support costs. The loss of electrical power on manned space stations can be a life-threatening situation due to the hostile environment of space. Such critical power systems (those to which power flow must not be interrupted), unlike other typical systems, are designed from the perspective of path redundancy, for increased reliability of operation.

Control of the distribution of electrical power has traditionally been carried out using a number of electromechanical relays which monitor the current conditions at specific points in the circuit. Recent advances in microprocessor technology have led to their use in circuit protection applications. Their inherent advantages are their reliability and speed of operation. However, a more significant advantage of using microprocessors to control a critical power application is that a single device can be used to oversee an entire critical power distribution system.

The requirements of the space station power control system must build upon existing analytic tools such as state estimators, bad data detectors, security analyzers, and power flow analyzer, to monitor and control electrical power. Virtually all researchers have concluded that the demands for space station power controller autonomy requires an integration of these tools with artificial intelligence techniques.

The main objective of an Intelligent Critical Power Controller (ICPC) is to quickly, intelligently, and reliably control the distribution of electrical power in space station installations. The controller is envisioned as working by modeling the power system on a continuous basis using the measured input values, and calculating output values, such as power flow currents and voltages at the outputs. These output values will then be compared to their actual measured values. Any discrepancy will cause a diagnostic activity to be placed into operation to determine the cause of the discrepancy, and initializing action to restore power flow to any critical load that may be affected. The controller will also contain a load scheduler in order to generate a load sequence which maximizes resource utilization without jeopardizing security.

This paper is a progress report on funded research investigating the use of model-based AI techniques in the creation of a power system controller for the manned space station. This research has two foci: first, the investigation into a model based solution to representing the requisite knowledge for such a controller; and secondly, an investigation into the real time issues confronting such a system. This investigation, which is at an early stage of development, has thus far focused on the creation of the knowledge base of devices and behavior required for a consistent and comprehensive model of the space station power system. Consequently, this is the major focus of this paper. In the next section, we sketch an overall working design of the entire ICPC. This is followed, in section three, by a description of the knowledge base.

2. ICPC DESIGN
A schematic diagram of a simple electrical power system is illustrated in Figure 1. The system is viewed as consisting of power sources (in the case of the space station, solar panels and batteries), a power distribution network of conductors and switches, and a set of loads. Currently, the authors are using this design to build a simulation model in software. Eventually, research plans include incorporating and testing the model by using a breadboard, such as the one currently being developed at Marshall Space Flight Center [17].

Proceedings of the 3rd Florida Artificial Intelligence Research Symposium, 1990, by the Florida AI Research Society

c. 1990 FLAIRS 0-96020-1732-5/90/00390-0189
A diagram of an automated power management controller is displayed in Figure 2. The controller is in charge of the control of the power system. The power management controller contains three lower-level assemblies: the power source controller, controlling the power generation and storage sources, the main bus switching assembly, protecting the electrical bus, and the power distribution assembly, controlling the distribution network [2]. It is essential to incorporate a preemptive, priority based handling of different types of faults so that the cases of multiple faults are managed properly [4]. For example, a dead short circuit on any of the system busses should be cleared before analyzing the relatively minor problem of marginal overloads of a particular load.

The system will operate in the foreground most of the time, assisting a scheduler in its task of scheduling loads [8]. The system relies on its knowledge that allows detection and classification of system operation from quantitative data to signal whether a threshold has been exceeded. Faults in the components of the power system are described by providing a set of models describing their behavior under a number of different fault assumptions [1]. Occurrences of any abnormal condition are identified by a continuous monitoring of all the system variables at the power generation as well as at the utilization end. The current model of the system is used as a reference while measuring the instantaneous values of the system parameters. The causes of the problem are identified using the knowledge captured in the knowledge bases. Hypotheses regarding the cause of the fault are generated and expected values tested throughout the system. The system will solicit additional data if needed [10]. The detection component of the system would classify each piece of data as normal, warning, alarming, or emergency, depending on its variance from the expected value of the parameter, as well as classifying the type of fault.

The recovery procedure consists in a correction generation module which consults the model in order to simulate a fix to the diagnosed fault. The reconfiguration generated by the correction activator is sent to the controller. Depending on the severity of the problem, human interaction may be required [10].

![Figure 2](image2.png)

**Figure 2: An automatic scheduler/diagnostic controller**

## 3. MODEL-BASED APPROACH

To realize the demands for a semi-autonomous intelligent electrical power controller, the authors have found the need for:

1. An integration of current analytic tools such as state descriptors with a knowledge-based control.
2. A continuous, real-time supervisory system.
3. A system that incorporates a wide range and variety of knowledge sources, involving planning and diagnosis.
4. An AI approach which incorporates quantitative, and possibly qualitative, knowledge.

A model-based system allows for capturing knowledge at many levels of detail, and allows for a variety of points of view. For example, the supervisory component of the system (one that controls the execution of the scheduler and the diagnostic components, for example), can possess a coarse-grained knowledge of how the system operates. It should be familiar with the possible states of the system (normal or emergency, etc.), and which procedures should operate in each state. It need not know the details of the operations performed by these procedures. Thus, the expertise required for overall load management can be depicted as a collection of states, and a set of transitions between these states to other states. This knowledge is depicted schematically in Figure 3. (5)]. The four operating states are identified as Normal, Preventive, Emergency, and Restorative. The graph-like representation has transition arcs representing the results of the state-of-health analyst. For example, a violation of a power constraint may lead from a normal or preventive state to an emergency
state. This transition is represented as a V-arc between states. The other types of transition include:

1. S: The system has been diagnosed as secure.
2. I: The system has been diagnosed as insecure.
3. C: The controller has taken adequate corrective measures.
4. R,R2: The controller has taken adequate restorative measures to correct an emergency situation.

In addition, each state has an associated set of continuous actions accompanying it. For example, in the normal state the tasks consist of dispatching power and monitoring the risks to system security. Thus, this state will imply the use of the power controller and state-of-health analyst. We thus view the overall actions of the system to fall into three general categories: those which are applied to operate the system under normal conditions, those which are applied to monitor the system’s operations, and those which arise for corrective or restorative reasons.

The knowledge base should contain information about the electrical system in a form that can provide an informative description of the various components as well as their functionality. All the components are represented as knowledge objects with attributes and links to other objects, as well as a model of their behavior. The description of a component will provide the information about a component’s location in the electrical circuit and to which of the other components it is connected. The functional description contained in some of the slots in the object frames will provide information as to how the object behaves.

For example, in the case of an object representing a solar panel, the information contained in some of its slots will describe how much electrical energy could be produced by that panel under a certain quantity of solar irradiation given other operating conditions.

A typical component frame will have a nomenclature slot which describes a component as a relay, solar panel, cable, circuit breaker, battery etc. In addition, the object frame will contain slots such as units, value, tolerance, time delay etc. depending on what kind of object the frame represents [12]. A summary of these attributes is provided below.

There are four slots used to describe a component’s relationship to other components. These slots represent the methods used by engineers for troubleshooting networks of related components. One of the slots describes the source of power to that object. Every object is considered to derive power from its source. The source of power could be of various types such as electrical, mechanical, chemical, hydraulic etc. For each object or component, there is a path through which this source of power could be reached. In other words the slot source path describes what other objects should be active or inactive in order for a component to be able to receive its energy from its source.

Here is a summary of some of the attributes contained in component objects. These are in addition to the familiar AIO (an instance of) and AKO (a kind of) slots typical of hierarchical or network-based representations:

- **SOURCE**: Describes the source of input energy.
- **SOURCE-PATH**: A boolean expression that describes how the upstream objects are connected to this frame.
- **STATUS**: An algebraic expression known as a transfer function. Required for analog devices only. Used to calculate the theoretical output value for a given set of input parameters.
- **CVALUE**: A value currently obtained from a measurement obtained from a sensor or a value inserted by a control command.
- **DELAY**: A value showing how many 1/80 parts of a second it will take for this object to change states. The delay inserted in this slot corresponds to the time constant of a given object.
- **SINKS**: A list of other objects which derive power from the object represented by this frame.

4. THE EQUIPMENT KNOWLEDGE BASE

A prototype of the ICPC is being implemented in software on the Knowledge-Based Autonomous Test Engineer (KATE), developed at NASA-RSC. KATE was chosen because it possesses many of the capabilities (in particular, a model-based architecture) deemed important to solve the problem of autonomous power control. We envisage the eventual need for modifications of KATE's design (e.g., for distribution of knowledge and parallel control). Our goal in using KATE initially is to have a working, consistent model of the system to test and evaluate. In this section, we discuss the knowledge base currently being developed using KATE.
5. **SUMMARY**

This paper has reported on the progress of research devoted to automate the control and distribution of electrical power on the manned space station using AI techniques. The overall goal of this research is to design a system autonomous enough to free the crew of most of the routine electrical diagnostic and monitoring activities, and yet accessible enough to allow for intelligent and efficient human intervention when needed. Thus, future research must include a consideration of the user interface to the ICPC. The primary goal of the initial research described in this paper is the improvement of the performance of diagnostic reasoning tools to meet the requirements of a space station power system controller. This paper has described the first steps in this direction.

6. **NOTE**

This work is being supported by a grant from the Florida High Technology and Industry Council.

7. **REFERENCES**


A KNOWLEDGE-BASED WRITER’S AID FOR SIMPLIFIED ENGLISH

Inen Syu, Sheau D. Lang
Department of Computer Science
University of Central Florida
Orlando, Florida 32816

Peter J. Kiscad
Institute For Simulation and Training
University of Central Florida
Orlando, Florida 32816
Margaret Thomas
Department of Psychology
University of Central Florida
Orlando, Florida 32816

ABSTRACT

In this paper we discuss the development of SEAN (Simplified English Analyzer), a knowledge-based writer’s aid for authoring training materials written in controlled English, such as Simplified English (SE). SE was developed as a subset of “Standard English” for writing procedural instructions for readers whose first language is not English. The SE writing rules limit the vocabulary, standardize grammatical constructions, and assign clearly defined meanings to approved words. As a result, users can easily learn SE and read manuals written in SE.

SEAN was developed as a prototype, automated writer’s aid capable of processing the text based on the SE writing rules and proving feedback to the writer for rule violations. SEAN uses a knowledge-based approach which represents the SE writing rules as Prolog rules incorporating an ATN grammar for sentence parsing.

INTRODUCTION

It has been a common practice to use English write technical documents, such as maintenance manuals and training materials, which are distributed for word-wise use. However, due to a variety of causes including the difficulty of the content and the difficulty of the writing style, the readability problems are the primary concern of the reader whose first language is not English. Over the years, world-wide companies or organizations have tried to solve the readability problems by preparing their technical documentation in some form of controlled English vocabulary, such as Simplified English (SE). SE is a result of joint effort between the Aerospace Industries Association (AIA) and the Association Européenne des Constructeurs de Matériel Aérospatial (AECMA). The vocabulary of SE consists of approximately 1500 words. Each word has a single, well-defined meaning. In conjunction with the limited vocabulary, SE is generated in accordance with a set of SE writing rules which define allowable grammatical constructions. The list of the SE writing rules is shown in the Appendix. To date, over 200 manufacturers were using the SE document for the preparation of aircraft maintenance manuals. In the military, the Air Force Human Resource laboratory has also recommended that SE be used to prepare all future Air Force manuals (Kaufin et al. 1989).

Texts written in SE are easy to read; however, they are very difficult to write. Writers must learn to restrict usage of syntax and vocabulary to that prescribed by SE and follow many writing rules. Several writer’s aid packages are available on the market, these include Grammatik III, Right Writer and Word Plus. They are capable of doing spelling check, word frequency calculation and writing style assessment. However, they either have no parsing capability, or their parsing capability is limited. Two more advanced system, EPISTLE and Editor’s Assistant which are implemented in main frames, use Natural Language Processing (NLP) techniques to achieve sophisticated text processing. EPISTLE includes a parser which is capable of diagnosing stylistic problems and handling most of the common grammatical errors. The Editor’s Assistant is a rule-based system which helps a copy editor in massaging a text to conform to a house style (Dale and Lid 1989).

SEAN is a Prolog based automated writer’s aid capable of processing the text based on the SE writing rules and proving feedback to the writer for rule violations. SEAN uses a knowledge-based approach which combines the use of production rules and an ATN grammar, implemented in Prolog at various levels. In the remainder of the paper we describe SEAN’s system architecture and its user interface and portability. We also report its current status and the result of preliminary testing as a writer’s aid.

KNOWLEDGE-BASED IMPLEMENTATION

SEAN is developed using Prolog. Prolog is a logical programming language which supports programming in modules and offers the advantages of problem isolation, parallel tasks programming, and ease of maintenance (Shafer 1987). These Prolog facilities and the ease of implementing production rules systems help the construction of knowledge-based systems (Cone and Rossi 1989). SEAN is structured in three modules: the rule base module, the database management module, and the user interface module. The modular design of SEAN also contributes to its portability and extensibility (Syu and Driscoll 1989).

Rule Base Module

The capability of checking text for adherence to SE writing rules and grammar rules is implemented in the rule base module. In the rule base module, the SE writing rules and ATN grammar rules are represented using the special rule syntax of Prolog. A Prolog rule has three parts: the HEAD, the BODY and the rule symbol (:-) (Borland 1988). The following shows the generic syntax of a Prolog rule:

```
HEAD :- <Subgoal>,<Subgoal>,...,<Subgoal>.

BODY
```

Prolog is a backward chaining system, that is, the above syntax can be described as "HEAD is true if BODY is true". The BODY of the rule consists of one or more Subgoals. Each Subgoal is a call to another Prolog rule. As the program makes the call, Prolog tests the called rule to see if it can be proven true. Once the current rule has been satisfied (proven true), the call returns, and processing continues with the next Subgoal. When all the subgoals have been satisfied, the rule is said to succeed. SEAN uses the depth-first search as the searching strategy. Due to
Prolog's built-in backtracking function, when a sentence is being analyzed, the Prolog system automatically searches the rule base until either it is forced to return all possible rules are checked. The example in Fig. 1 illustrates the representation of a grammar rule and a SE writing rule using the backward chaining strategy.

Sample grammar rule:
Sentence = Noun-phrase Verb-phrase | Verb-phrase
Backward chaining representation:
string_is_sentence if
  string_is_noun_followed_by_verb
or string_is_verb
Sample SE writing rule:
Rule(1.1.2) Only use allowed SE words.
Backward chaining representation:
violate_rule_1.1.2 if
  word_of_sentence_not_in_SE_dict

Fig. 1 Sample Backward chaining rule representation.

To conclude whether a sentence has any rule violation, we must check if the sentence matches the constraints given.

a. Does the sentence have correct syntax?
b. Is each word of the sentence a SE word?

Fig. 2 shows a sample scenario using the rules in Fig. 1 to detect SE rule violations.

Sample sentence:
The fueling control panel is turned off.

Prolog rules:
sentence(STRING, NP, VP):-
noun(X, STRING, NP),
verb(Y, STRING, VP),
rule112(X, STRING),
sentence(Y, STRING, VP),
check(NP, VP).

Output Message: Don't use "turned". Replace with "switched".

Fig. 2 A sample scenario of detecting SE rule violations.

The two sample rules in Fig. 1 are translated into Prolog rules (predicates) "sentence" and "rule112" in Fig. 2. STRING is a sentence read from user's text. The "rule112" predicate invokes two predicates: "sentence" and "check". The "sentence" predicate detects grammar errors of the sentence. After the sentence is parsed into a noun phrase (NP) and a verb phrase (VP), the SE writing rules are checked. The "check" predicate examines the usage of words. Eventually, an output message is generated showing the detected SE rule violation that "turned" is not a SE word and it should be replaced with "switched".

There are 45 SE writing rules (see Appendix). About a third of the rules cannot be implemented for reasons such as ambiguity, lack of consideration of automation, etc. However, near twenty of the rules have been automated in SEAN and the rest are now in progress. The rules currently being incorporated in SEAN include:
* Use approved nomenclature
* Use consistent names for parts
* Break up noun cluster
* Use active voice
* Use imperative form in procedures
* Rules relating to sentence and paragraph structure

The violations of the above writing rules can be isolated by analyzing the syntax of a text. In current SEAN, the syntax of a text is parsed by an ATN grammar (Bates, Benet and Newman 1978) which consists of fifty states and transitions. The SEAN parsing routines translate the texts and operations of the states and transitions into grammar rules.

Database Management Module
The tasks of the database management module are loading the dictionary, searching words, scanning user's texts, and producing output files. The current SEAN dictionary contains over five thousand words divided into general entries and domain specific entries. General entries are words such as auxiliary verbs and prepositions which can be applied to any domain. Specific entries are nouns, verbs, adjectives, adverbs, and the alternative words for those that are not SE allowed words. The database management module loads the dictionary into a memory-resident database in order to speed up searching. This approach requires storing the dictionary in main memory, in a compact and efficient way. Other techniques for efficient database access from Prolong can be found in (Ceri, Gelfand and Wiederhold 1989).

User Interface Module
SEAN can be used to process texts in an interactive mode or batch mode. In the interactive mode, the monitor screen is divided into two windows: the sentence being processed is shown in the upper window, the corresponding feedback messages, which are generated according to the results of the sentence parsing and SE writing rule checking, are displayed in the lower window. An output file is also generated which contains the processed text marked up by the violations of SE writing rules and ATN grammar rules.

EXPERIMENTAL STUDIES
Studies of the preliminary testing of SEAN as a writer's aid have been conducted by twelve technical writing students at the University of Central Florida. This study group had been given training in the application of SE writing rules. In the first study, each student was given 44 sentences to review and asked to mark on each sentence any violation of SE rules. The percentage of failures to detect rule violation were as follows: required commas, 72%; mandatory imperative, 64%; sentence too long, 37%; passive voice, 29%; omission of an article, 27%; multiple instructions in same sentence, 26%; incorrect verb form, 21%; and noun clusters, 8%. These results show that even provided with a copy of the SE rules these writers still overlooked violations with regularity.

The same group of students also examined the utilities of SEAN. They were given 10 brief paragraphs. Having access to the SEAN's utilities resulted in an important time savings. The average amount of time required to edit the paragraphs without SEAN's utilities was about 55 minutes. In comparison, when SEAN assistance was given, the average time required was 43 minutes. This represents a time saving of nearly 22%.

These studies have demonstrated that writing SE materials without automated feedback is difficult, even for experienced technical writers. The use of the SEAN software saves time and results in increased compliance with SE vocabulary and writing rules.

CONCLUSION
In this paper we discussed the development of SEAN, a knowledge-based writer's aid for Simplified English. Preliminary findings in controlled field tests are showing the usefulness of our authoring aids. However, not all violations of SE writing rules can be detected by merely checking the syntax of the text. Deeper
understanding the text, for example, using the case grammars for determining the thematic roles of words in the text (Winston 1984), need to be used in certain SE writing rules. This work is currently being investigated.

ACKNOWLEDGEMENT

The research reported here has been encouraged and supported by the Florida High Technology and Industry Council.

APPENDIX

SE Writing Rules:
1.1.1 Choose the words for procedures from SE dictionary.
1.1.2 Use SE approved words as the part of speech.
1.1.2.2 Keep to the proper meaning.
1.1.3 You can use words that are "technical Name".
1.1.4 You can use verbs that are "Manufacturing Processes".
1.1.6 Make your instruction as specific as possible.
1.2.1 Write the group of words easy for reader to understand.
1.2.2 Break up noun clusters which have more than three nouns.
1.2.3 Noun cluster can be used for Technical Names.
1.2.4 If it is possible, put an article or a demonstrative adjective before a noun.
1.3.1 You can use the following tenses of the verb: present tense, the past tense, the simple future tense.
1.3.2 Use the active voice.
1.3.2.1 Test the passive voice.
1.3.2.2 Change the passive voice into active voice.
1.3.3 Identify verbs or nouns.
1.3.3.1 Let the verb show the action.
1.3.3.2 Do not omit a verb to make your sentences shorter.
1.4.1 Keep the texts as simple and readable as possible.
1.4.2 Keep sentence as short as possible
   - Procedure: The maximum length is 20 words.
   - Description topic: The maximum length is 25 words.
1.4.3 Write short sentence.
1.4.3.1 Keep to one topic per sentence.
1.4.3.2 Use connecting words to join separate sentences.
1.4.3.4 Use tabular layout for complex texts.
1.4.4 In descriptive writing, try to vary sentence length and constructions to keep the text interesting.
1.5.1 Write only one instruction per sentence.
1.5.2 Use the tabular layout to show relative steps.
1.5.3 Write more than one instruction per sentence only when two actions have to be done at the same time.
1.5.4 If you write an instruction with a descriptive statement, the instruction must be separated from the statement by a comma.
1.6.1 Use paragraphs to show the logic of the text.
1.6.2 Each paragraph can have only one topic.
1.6.3 Always start the new paragraph with new topic sentence.
1.6.3.2 After the topic sentence, the remaining sentences must go on to develop the topic.
1.6.3.3 Use connecting words to make the relationship between sentences and paragraphs clear.
1.6.4 The maximum length of a paragraph is 6 sentences.
1.6.4.1 Do not use one-sentence paragraph more than once every 10 paragraphs.
1.6.5 Present new and complex data slowly.
1.7.1 A warning or a caution tells the technician that the parts of the procedures can be dangerous.
1.7.3 Add a brief explanation to a warning or a caution.
1.8.2 When you count sentence length, the colon and the dash count as a full stop.
1.8.2.1 Use the colon and dash only in tabular layout.
1.8.3 Use the hyphen as a joining signal.
1.8.3.2 Word in a hyphenated group counts as a separate word unless it is a prefix.
1.8.3.3 In a technical name that consists of four or more words put hyphens between words.
1.8.4 Text inside parentheses counts as a new sentence.

REFERENCE


VALIDATION OF EXPERT SYSTEMS

Suzanne Smith
Department of Computer and Information Sciences
East Tennessee State University
Johnson City, Tennessee 37614

Abraham Kandel
Department of Computer Science
Florida State University
Tallahassee, Florida 32306

ABSTRACT

The need for effective tools and techniques for validating expert systems is addressed in this paper. The validation techniques of the traditional software life cycle and the current practices in the validation of expert systems are examined. A tool to support the validation of expert systems is presented.

STATEMENT OF PROBLEM

The reliability of software is a key issue in industry today. Engineers of conventional software have developed and refined methodologies to ensure this reliability through the use of verification and validation techniques in the traditional software life cycle. The widespread acceptance of the software life cycle approach and the methodologies applied within it has led to the production of reliable software products.

As expert system technology moves from the research lab into industry, this same software reliability is expected. However, widely accepted means for providing verification and validation in expert systems have not been developed at this time. The acceptance of expert systems in industry today is greatly hampered by the inability to show that expert systems are correct.

TRADITIONAL SOFTWARE LIFE CYCLE APPROACH

Terminology

The activities of verifying and validating software are so closely tied

that many methodologies approach verification and validation as one task called V&V. The activities of V&V are formal tests of the software which work together toward determining whether the software performs its intended functions and ensuring the quality of the software (Deutsch 1988). Verification and validation are part of a total software development methodology which is conceptualized in the software engineering life cycle approach. The life cycle approach contains phases for problem definition (i.e., systems analysis and requirements analysis), development (i.e., software design, coding, and software testing), and maintenance. V&V activities are present throughout the life cycle phases and are integrated with other development activities in order to ensure the reliability and quality of the software product. Because of the close association of these activities, the boundaries of their definitions are often blurred in practice and literature. A clear definition of verification, validation, and related activities is a preliminary step to understanding validation.

Verification has been defined as the activities which answer the following question: "Am I building the product right?" while validation answers "Am I building the right product?" (Boehm 1984). Another view of verification is the ensuring that the software is without errors. Activities related to this definition include unit testing and design unit testing. Verification also involves ensuring that the product of each phase of the software life cycle correctly implements the specifications of the previous phase in the life cycle. For example, the design specification, which is the end product of the design phase, must correctly implement the intentions of the Software Requirements Specification, which is the end product of the analysis phase, and the source code must
correctly implement the intentions of the design specification. Formal proofs of correctness are sometimes used to demonstrate that the source code correctly implements the design specification (Fairley 1985).

Other software engineering activities which are closely related to V&V and, therefore, are often confused with validation include testing and evaluation. Testing is an aspect within verification and validation. It helps in determining if a software product meets its requirements. There are many different types of testing such as unit testing, design unit testing, and system testing. Certain types of testing are used to perform the validation of software. These types are described in the next section. Validation and evaluation are often confused because they both are concerned with the accuracy of the software and a product. Evaluation, however, differs from validation in that it is also concerned with the utility of the software product. While criteria for validation are completeness and consistency, criteria for the evaluation of software include portability, efficiency, human-engineering, modifiability, and cost-effectiveness (Liebowitz 1986).

Validation is concerned with demonstrating whether or not the software product actually solves the user’s problem. According to the IEEE definition, validation is “the process of evaluating the software at the end of the software development process to determine compliance with the requirements” (Fairley 1985). These requirements have been previously defined in the analysis phase of the software life cycle and are documented in the end product of that phase, the Software Requirements Specification. This document includes a section of criteria by which the software product is validated. This section describes the functions and features which the final software product must include. The validation process seeks to demonstrate the presence or absence of these functions and features in the software. Validation testing is performed with the use of test cases. The ultimate goal of the validation process is the obtainment of the confidence of the user that his/her software product correctly does what it is supposed to do.

Validation Activities in the Traditional Software Life Cycle

In the traditional software life cycle, validation is one of the final activities before the release of the software product. Validation occurs within the testing phase of the software life cycle. Within this testing phase, the four major testing activities are unit testing, design unit testing, system testing, and acceptance testing. The first two activities are part of the verification process for the software. The last two activities, system testing and acceptance testing, combine to perform the validation of the software.

Both system testing and acceptance testing are “concerned with ascertaining that a software system meets user requirements” (Ince 1985). System testing, like unit testing and design unit testing, is still done in the “laboratory environment” (i.e., not in the final operational environment). Acceptance testing, however, is performed in the operational environment of the software. The operational environment combines the software product being validated with the other software and hardware which comprise the whole system of which the software product is a part.

Validation testing (i.e., system testing and acceptance testing) is most commonly the execution of a series of test cases designed to demonstrate the presence or absence of the functions and features specified in the validation criteria in the Software Requirements Specification document. The generation and maintenance of these test cases are an important part of the validation process.

These test cases are applied in a black box testing approach. Black box testing is concerned only with the inputs to and outputs from the execution of test cases. Black box testing is used in validation because it focuses on functional requirements. The results (i.e., the outputs) from these test cases are then compared to the known results.

DIFFERENCES BETWEEN TRADITIONAL SOFTWARE AND EXPERT SYSTEMS

Expert systems have some unique characteristics which differentiates them from non-expert system software (i.e., traditional or conventional software). It is important to understand these differences and how they impact the validation process. Expert systems are normally developed using an evolutionary or prototyping approach to software development instead of the traditional life cycle approach. Rarely do developers of expert systems produce specification documents which delineate all the requirements for the expert system before they begin working on the design of that expert system.
Normally, as expert systems are designed and prototypes built, new requirements are realized and incorporated into the design and prototype. This evolutionary approach also does not normally include a detailed list of the validation criteria by which the resulting expert system is to be validated. Without validation criteria, the validation process is a matter of guesswork.

Expert systems differ from traditional software in that expert systems embody knowledge which is to be used to emulate human intelligence. The content of the knowledge base is normally developed in the same evolutionary manner as the rest of the expert system and, therefore, is constantly changing and growing during the development and operational phases. In addition, this knowledge may represent uncertain and/or imprecise information. Validation of this knowledge is as important as the validation of the overall expert system. As the knowledge base grows or changes during the lifetime of the expert system, it is also important to revalidate the knowledge base and the expert system. These validation concerns are not issues in traditional software.

For traditional software, the results produced during validation testing are easily validated because they are exact in their correctness (i.e., the results are either correct or incorrect). It is harder to judge the results of an expert system in such an exact manner. Validating an expert system is to validating traditional software as "grading an essay examination is to grading a true-false examination" (Green and Keyes 1987). Not only do expert systems often produce complex results, but it is also difficult to have human experts agree upon what should be the correct result. There may be several acceptable results for a particular problem or no agreement among several experts on what the acceptable result actually is. Expectations on the level of expertise to be demonstrated by an expert system also differ among human experts. All these factors add to the complexity of validating expert systems.

**EXPERT SYSTEMS APPROACH**

**Terminology**

The basic definition of validation is the same for expert systems as for traditional software. Validation is still concerned with demonstrating whether or not the software product actually solves the user's problem.

Validation for an expert system generally focuses on whether the result produced by the expert system is the expected result for that situation. In other words, validation for an expert system seeks to answer the question "How does the system perform when compared to an expert?" (Bielański and Lewand 1988).

The differences between expert systems and traditional software, as previously discussed, require that validation for an expert system deal with issues which are not concerns in traditional software development. For example, both the decision-making processes and the knowledge base in an expert system need to be validated. To accomplish this requires:

1. ascertaining what the system knows, does not know, or knows incorrectly
2. ascertaining the level of expertise of the system,
3. determining if the system is based on a theory for decision making in the particular domain, and
4. determining the reliability of the system (O'Leary 1987).

The accuracy of the knowledge within the knowledge base and the accuracy of the conclusions or advice of the expert system are addressed through the examination of these aspects of the expert system. All of this works together to instill the user with the confidence that the expert system is doing what it is supposed to do.

**Current Practice in Validation of Expert Systems**

Typically, expert systems are validated by manually applying a set of test cases to the system and comparing the system's results against conclusions of a human expert on the same test cases. The expert system is considered valid if it matches the human expert's conclusions on a predetermined percentage of test cases. This process is often referred to as the Turing test. The validation process normally occurs after the development of the knowledge base. The validation process is often done in an ad hoc or informal fashion (O'Reefe et al. 1987). In many expert systems, the availability of adequate test cases is limited. Although systems such as CASNET, MYCIN, and PROSPECTOR were validated using selected test cases, the number of test cases applied during validation was very small (Politakis 1985).
While Turing tests are normally used to validate the performance and/or reasoning of expert systems, the knowledge base can also be validated through Turing tests. Test cases to exercise certain aspects of the knowledge base are selected for the validation process. Another method of validating the knowledge base is for the expert to manually review its contents in order to determine its accuracy. This method is cumbersome with large knowledge bases.

The specification of validation criteria during the initial phases of expert system development is rare. There are several reasons why the requirements and, therefore, the validation criteria are not formally defined in current practice. Often times in the prototyping approach to developing expert systems, as described earlier, the requirements evolve during the building and evaluating of the prototype. Another reason for not formally defining requirements and validation criteria is that the developer or user is not willing to commit the time and effort to producing and maintaining a requirements specification document. Some developers do not anticipate performing a formal validation test on their expert system and, therefore, do not need to determine validation criteria (Green and Kayes 1987).

CASE VALIDATOR

The examination of validation strategies and methods in the traditional software life cycle and current practice in expert system technology reveals major areas of validation which need to be addressed in any strategies and methods for validating expert systems. Such strategies and methods need to aid in the execution and maintenance of test cases and to enhance validation coverage. The tool presented here, called the case validator, supports these strategies and methods. This tool will facilitate the use of test cases in the validation process through:

1. dynamic repository of test cases,
2. automated record keeping of validation results for current and past validation, and
3. analysis of the effectiveness of validation test cases in relation to the knowledge base.

Repository of Test Cases

The case validator will provide the knowledge engineer or expert system developer with a means of storing a repository of test cases. The entry of test cases will be facilitated by a user-friendly entry program provided within the case validator tool. The repository will be expandable at any time during the development or operation of the expert system. The execution of part or all of this repository can be performed in batch mode. For each test case, the case validator tool will also store the final conclusions where they are already known and a list of facts and certainties related to a particular test case where available.

Record-keeping Facility

The automated record-keeping facility within the case validator tool will provide a means of keeping track of the results of the execution of the test cases in the repository. This facility will display in a tabular report the expert system's conclusions along with the conclusions of a human expert which are stored within the test case repository. Results from previous test case executions will be maintained by the case validator to provide a history of the performance by the expert system (Hall et al. 1987).

Coverage Analyzer of the Knowledge Base

The coverage analyzer of the knowledge base will aid the expert system developer and knowledge engineer in validating the contents within the knowledge base. This feature will provide information for use in determining the effectiveness of the test cases in relation to the contents of the knowledge base. Such information can provide a measure of the completeness of the validation coverage (Oliver 1987). The coverage analyzer of the knowledge base will also provide suggestions on how to improve the validation coverage of the knowledge base. These suggestions are based on both domain-dependent knowledge and domain-independent validation knowledge.

CONCLUSIONS

For expert systems to achieve widespread use in industry and government, expert system technology must establish the means of gaining confidence in the reliability and quality of the decisions made by expert systems. Tools and/or methods to enhance and facilitate the validation of expert
systems are a way to increase this confidence. The strategies and methods proposed in this paper will provide the expert system developer with a systematic means of validating the expert system and with conveying the results of such validation. Such validation capabilities, together with an overall methodology for verification and validation, are necessary for establishing the quality and reliability that industry now requires of expert systems.

REFERENCES


EXPERT SYSTEM VALIDATION AS IT APPLIES TO EXPERT SYSTEMS USING FRAME-BASED KNOWLEDGE REPRESENTATION

Ana Maria Cheng
Honeywell Inc., Military Avionics Division
11601 Roosevelt Boulevard
St. Petersburg, FL 33716-2202

and

Lawrence O. Hall
Dept. of Computer Science
University of South Florida
Tampa, FL 33620

ABSTRACT

This paper addresses the problem of Expert System Validation. Its purpose is to expose the problem as an important concern in the field of Expert System research, to describe the evolution of expert system validation, and to arrive at a methodology for validating a particular set of expert systems not yet addressed by the current literature. Namely, the set of frame-based expert systems.

EXPERT SYSTEM VALIDATION

Expert system validation is defined as demonstrating that the expert system performs the functions it is required to perform (e.g. diagnose, predict, instruct, explain, etc) and is usable for the intended purposes. Because of the seriousness of some expert system applications, these functions must be performed correctly without the undue danger of reaching erroneous or illogical conclusions that may lead to disastrous effects.

Knowledge engineers say that expert systems perform at levels close to those of human experts. Some do but a lot of them don’t. The public can not trust an expert system to make critical decisions until this gap in performance is considered non-existent. One reason for this problem is that expert systems are not subject to strict, formal validation until the latter stage in their development, the testing stage. That is, an expert system’s performance is typically validated by running the finished expert system, using test cases as input, and comparing its results against known results obtained from the expert. If these don’t match, the validating engineers tend to use subjective analysis to explain the contradiction. This may seem like a simple approach but finding the cause of the failure and correcting it can be a difficult and expensive task.

If expert systems are to be used on a regular basis, they must be carefully validated especially for critical applications. Expert systems researchers recognize validation as an important concern and are now proposing the development of formal validation methods.

PROBLEMS IN EXPERT SYSTEM VALIDATION

Although validation methods have been developed to be applied to regular procedural software, there exist a set of problems faced when applying them to expert systems. Some are due to the separation of inference from knowledge and the non-procedural nature of expert systems. Some of these problems are:

1. Having vague objectives or ever-changing requirements.
2. The fact that the resulting code of an expert system does not resemble the actual execution sequence, makes the possible conclusion states difficult to predict.
3. It is difficult to obtain an ideal set of domain-representative test cases which may test all states and possibly represent the domain.
4. There are also questions as to what part of the system to validate and when to validate. Also as to what to validate against: known results or expert performance. The consensus is to validate the inference engine, knowledge acquisition facility and explanation facility early. The performance of the system depends mostly on the validity of the knowledge, thus validation of the knowledge must begin when sufficient knowledge is present in the knowledge base so that complete reasoning paths may be tested. This process must continue through the life of the system when knowledge is modified or added. It would be ideal to always validate against expert performance, but he/she is seldom available. For this reason, test cases with known results are used.
5. Knowledge representation plays an important role in the task of validation. In attempts to achieve greater representational power, several methods of representing knowledge are being developed as a result of the ongoing research in the area. Unfortunately these efforts are not equaled by those made in the field of expert system validation. The fact the knowledge representation schemes continue to emerge, while validation of such knowledge is neglected, is an important factor behind the lack of public belief in expert systems.
6. Methods for representing uncertainty are often misunderstood and therefore may be incompletely implemented and wrongly interpreted by a validation tool.
7. Finally, in many knowledge bases, domain-level knowledge, meta-level knowledge, and control knowledge, are combined using the same representation scheme, thus confusing the validation analysis process. This makes it necessary to incorporate into the validation tool a way of distinguishing the various types of knowledge in order to validate them separately.
APPROACHES TO EXPERT SYSTEM VALIDATION

It has been found that most, if not all of the work done on this subject concentrates on the validation of rule-based knowledge only. The findings and conclusions presented in this paper are based on a variety of primitive approaches, originating from regular procedural software validation methodologies, as well as more recent approaches with varying degrees of effectiveness. Two such projects are the development of SEEK (System for Empirical Experimentation with Expert Knowledge) (Poilatis '85), and EVA (Expert System Validation Associate) (Chang, et. al. '87, '88).

The SEEK System

SEEK is an interactive system that offers advice on rule refinement during the design of a diagnostic rule-based expert system model whose domain is Connective Tissue diseases. It can verify the model over an entire set of test cases, or one test case in particular, and suggest experiments for modifying the rules when the model's results and the known results don't match.

SEEK makes use of two knowledge sources: (1) expert knowledge in the form of rules, and (2) experience in the form of test cases with known conclusions. It simulates the inference process, and detects regularities in performance when cases are misdiagnosed.

The system suggests two classes of experiments: 1) Generalization or weakening of rules, thus making a rule easier to satisfy or making its impact on decision-making carry more weight. This can be done by removing clauses from the premise of the rule, or by increasing its confidence level, respectively, and 2) Specialization or strengthening of rules, thus making a rule more difficult to satisfy or making its impact on decision-making carry less weight. This is done by adding clauses to the premise, or by decreasing its confidence level, respectively.

In order to suggest these experiments, rule-refinement statistics such as the number of satisfied components in candidate rules, number of components needed for satisfaction, number of cases in which rules are used to reach the correct and incorrect conclusions, number of cases suggesting the generalization or specialization of a rule, etc. are gathered. These are then examined in the premises of a set of heuristic rules to determine and suggest the actual modification.

The EVA system

EVA is a meta-knowledge-based system whose purpose is to define and develop automated tools to validate the structural, logical, and semantic integrity of rule-based knowledge-based systems.

EVA has two component sets of modules. The first consists of conversion algorithms to translate the application (the knowledge of the expert system being validated) and its validation and control statements into a format that can be understood by EVA's second set of modules, the validation modules. This set includes the logic checker, structure checker, semantics checker, omission checker, behavior checker, and rule proposer modules. This latter module is similar to SEEK in that it examines the behavior of the rules during inference, and suggests the specialization or generalization of rules. EVA's modules detect errors such as inconsistency, numerical incompleteness, unreachable or dead-end rules, recursiveness, irrelevancy, errors in legal values for variables and data types, and omissions in the application knowledge.

Both SEEK and EVA, ease the task of knowledge base validation and correction through simulation and thorough analysis of the knowledge elements. Both efforts are aimed at rule-based knowledge validation, but present two different contexts for validation. SEEK validate the correctness of the expertise, requiring the support of test cases with known results, and a model of the inference process with additional statistics gathering and analysis capabilities. EVA validates the integrity of the knowledge with techniques that apply to the encoded knowledge only.

OUR IDEAL APPROACH TO GENERIC EXPERT SYSTEM VALIDATION

It would be desirable to extend facilities such as those of SEEK and EVA to expert systems of any type, being developed or existing. We could have a "generic" validation tool to build expert systems and validate them during their development as well as throughout their lives. If this tool is to be used on any expert system, then it is necessary to have translating programs to change the expert system's knowledge representation into the one used by our tool and back, since it is its own knowledge representation for which validation statements and refinement advice are generated. The tool should also be able to model the expert system's inference mechanism. These two notions, especially the latter one, seem somewhat infeasible considering that as the field of expert systems evolves, newer and more sophisticated methods of representing knowledge and of inferring on that knowledge continue to emerge but we are very far from finding a knowledge representation scheme which encompasses all expert systems. Such a knowledge representation scheme could be called the "generic" knowledge representation scheme. A logical step to take then, is to aim validation at a different set of expert systems from that already addressed, namely, the set of frame-based expert systems.

VALIDATION OF A FRAME-BASED EXPERT SYSTEM

In order to arrive at a methodology for validating frame-based knowledge, we used CENTAUR (Alkens '83), a diagnostic expert system in the pulmonary physiology domain which combines frames and rules in an effective manner, and aimed validation at its type of knowledge.

The frames in CENTAUR are called prototypes. They have production rules attached to their slots, used for determining attribute values and as control knowledge which may be seen as equivalent to the procedural attachments often associated with frames. There are five kinds of rules, conforming to the different types of knowledge (domain- and meta-level knowledge). They are grouped according to the situation in which they must be applied: 1) Inference, 2) Summary, 3) Triggering prototypes (antecedent), 4) Faci-residual, and 5) Refinement rules. Control knowledge as well as some domain-level knowledge is implicit in the prototype slots as well as the frame network. CENTAUR explicitly separates the different types of knowledge and this is a definite advantage during validation.
VALIDATING THE INTEGRITY OF CENTAUR'S FRAMEWORK

CENTAUR's meta-knowledge comes in the form of triggering rules, prototype control slots, and importance measures.

The triggering rules are antecedent rules used to trigger the disease prototypes. The control slots associated with the prototype contain Lisp function calls that are executed as the action part of a rule whose premise is "if the current situation matches the situation described by this prototype".

The importance measures (IM) are associated with some slots to represent how important the concept represented by the slot is in confirming the presence of the disease represented by the prototype.

To validate CENTAUR's meta-knowledge, the above three forms of it should be validated separately. The triggering rules may be validated using the techniques used and proven effective for rule-based knowledge. The control slots require a set of constraint statements for specifying which results should be allowed so that the results from executing the Lisp functions don't conflict. The obvious way to validate the importance measures requires the support of the expert for ranking the components within each prototype and assigning values to the importance measures.

VALIDATING THE INTEGRITY OF CENTAUR'S DOMAIN KNOWLEDGE

Recall that we are looking at the rule-based part of the domain knowledge first, and that the frame network will be addressed for validation separately. CENTAUR's rule-based domain knowledge comes in the form of the inference, summary, refinement, and fact-residual rules. They all address domain parameters to infer, summarize, refine, and account for expected or unexpected values and conclusions.

The validation of this part of CENTAUR's domain knowledge may be achieved using the same efficient techniques already developed for rules.

From our approach to "generic" validation and the two successful approaches of SEEK and EVA we obtain concepts which form the basis for the validation of a frame network.

Perhaps the most important concept is that of translating knowledge to a "generic" knowledge representation scheme for the generic validation of an expert system's knowledge. It was concluded that such a scheme does not yet exist and will not exist in the near future. Perhaps we can find a way of translating frames to a knowledge representation for which efficient validation techniques do exist, and once validated, translate the knowledge back to frames. It is clearly apparent that the "target" knowledge representation, although not generic, should be rules.

This problem is referred to as Knowledge Transformation. Knowledge engineers originally encountered it when faced with transferring an expert system from one shell to another, combining two expert systems, or establishing communication between expert systems. Peter Rothman [Rothman, 1988] divides the problem into six subproblems:

1. Combining two systems with the same problem domain.
2. Combining two systems with different problem domain.
3. Transforming knowledge from one knowledge representation scheme to a similar one.
4. Transforming knowledge from one knowledge representation scheme to a dissimilar one.
5. Altering knowledge within a single knowledge representation scheme to produce more time- or space-efficient knowledge.
6. Transforming knowledge between separate physical devices.

We are concerned with subproblem 4, namely, transforming frames to rules and rules to frames. Fortunately for us, most of the work on the subject has concentrated on the transformation between rules and frames. Bhanvi Thurasingham ['Thurasingham, 1989] proposes ways to achieve the above but the development of these strategies is by no means complete. There are numerous important factors that can not be overlooked. The major concern during transformation of knowledge representation schemes must be to preserve the knowledge in all its aspects. Rothman lists the following qualities as those which must be preserved:

1. Canonicity refers to maintaining the meaning of the knowledge. As an example, the statement LIKES(JOHN, CANCY) must retain the meaning that "John likes candy" and not that "candy likes John".
2. The truth values of an assertion are usually True or False. For incomplete or uncertain knowledge other values such as Possible, Unknown, and Undecided might be used. The meaning of these must be preserved.
3. For representing uncertainty, several techniques such as Bayesian probability theory, possibility and necessity sets, fuzzy logic, and others are used (Barr, et. al '81). Translating a representation that handles uncertainty requires that the association of certainty measures with knowledge and how they propagate during the inference process remain equivalent in the target representation.
4. Temporality refers to the points in time when truth values are assigned to assertions during the life of the system and with respect to chronological time.

5. Procedural knowledge refers to the preservation of the conditions that lead to executing a procedure as well as the preservation of its function and its outcome.

The above apply to all knowledge representation schemes and the translation process. A variety of additional conflicts are encountered when translating to and from frame-based knowledge representations, such as missing or excess slots, conflicting defaults, and inheritance conflicts.

Were it not for these problems, translating frames to rules, validating them, and translating them back to rules would provide the solution to validating frame-based systems such as CENTAUR. The peculiarities of frames and their domain-dependent structure make the goal of generic expert system validation even more difficult to achieve.

VALIDATING THE CORRECTNESS OF CENTAUR'S KNOWLEDGE

In the second context of validation, more involved processes are required. For this purpose, techniques based on SEEK's approach to validation and SVA's rule prover module should be used. First, a domain-representative set of test cases is needed. This requires the cooperation of a varied group of experts. In addition, a model of the inference engine is required. This model must perform the same inference strategy used by the actual system. It must also keep track of a set of figures calculated while observing the regularities that occur when the results from running the model on the test cases, and the results stored with the test cases don't match. These figures are then used to suggest the experiments required to arrive at the correct results. Some possible figures are: (1) the number of cases in which particular rules of each type are used to reach the correct and incorrect conclusions, (2) the number of cases in which the rules are used to select the correct and incorrect prototype to be examined (3) the number of satisfied clauses in triggering rules since they trigger disease prototypes and could be generalized or specialized in order to correct a test case, (4) measure of closeness to connecting the case, etc. Some of the suggested experiments may be: (1) the generalization or specialization of rules of all types, (2) changing importance measures, etc. It can be seen that these techniques are very dependent on the knowledge representation scheme used. Again, we see how far we are from the goal of generic expert system validation.

THE RESULTING METHODOLOGY

Provided there exists a transformation process between frames and rules in which knowledge is totally preserved, the resulting methodology for validating the knowledge of a system such as CENTAUR consists then of the following steps:

1. Validate for the integrity of the knowledge:
   A. Separate the knowledge into groups by its type and validate the frame network separately. The resulting groupings are:
      1. Meta-level Knowledge
      2. Domain-level Knowledge
      3. Frame Network Knowledge

   B. Validate Meta-level knowledge within the prototypes:
      1. Validate the set of triggering rules by performing logic checking, structure checking, omission checking, and behavior checking.
      2. Validate the control slots through the use of constraint statements to be applied to the Lisp functions called in each control slot and to specify the set of permissible/non-permissible outcomes from the execution of the Lisp functions.
      3. Validate the Importance Measures (IM) by ranking the components represented by the component slots within a prototype and assigning IM values.

   C. Validate Domain-level knowledge within the prototypes:
      1. Validate the set of Inference rules by performing logic checking, structure checking, semantics checking, omission checking, and behavior checking.
      2. Validate the set of summary rules using the above techniques.
      3. Similarly, validate the set of refinement rules.

   D. Validate the Prototype Network:
      1. Translate frames to rules.
      2. Apply above techniques to the rules.
      3. Translate validated rules back to frames.

II. Validate the correctness of the knowledge.

   A. Generate domain-representative set of test cases.
   B. Build expert system model and implement techniques for recording and computing statistical and historical information and suggesting experiments.
   C. Use the model on the test cases.

The above methodology is very much tailored to CENTAUR's knowledge representation scheme since it references the particular types of rules and the prototype constructs. The principles of it may be applied to other forms of frame-based knowledge. That is, to use the established techniques for any type of rule-based knowledge, and knowledge transformation for the frame network.

Regarding incremental validation, not all the steps above may be required. Validating for integrity, the methodology should be modified to be applied only to the new/modified knowledge along with any old affected knowledge. Validating for correctness, the expert system model should be run using the entire knowledgebase, but only on those test cases which reference the new and the old, affected knowledge. Some analysis of the knowledge base will be necessary in order to determine this.

Although not yet tested, this methodology's intent is to lay the groundwork for those interested in continuing the effort toward validating frame-based expert systems, as well as expert systems using knowledge representation schemes different from rules.
At present, expert system validation experience is very limited, especially for knowledge representation schemes other than rules. Effective methodologies will evolve only as the result of future collaborative efforts and critical evaluations of that experience. It is hoped that this paper has exposed the need for continuing the quest for solutions to the problems of knowledge representation, knowledge transformation, and most urgently, expert system validation.

REFERENCES


[Stefik] Mark Stefik, An Examination of a Frame-structured, Representation System, Computer Science Department, Stanford U., Stanford, California 94305.


AUTOMATIC PROGRAMMING FOR ARRAY MANIPULATION

Jie Wu
Department of Computer Engineering
Florida Atlantic University
Boca Raton, FL 33431

ABSTRACT

Automatic programming through deductive synthesis is discussed in this paper. Four general-purpose rules, based on Manna and Waldinger's work, are proposed by which a program is derived from a formal specification through manageable, controlled steps which guarantee that the final product meets the initial specification. The method proposed is applied to the SSS, a synthesis system for constructing array-manipulating program from a given specification.

1. INTRODUCTION

It is believed that the ultimate goal of artificial intelligence applied to software engineering is automatic programming (Rich and Waters 1986). However, it is unlikely that automatic programming for all possible domains will be achieved soon. Most work in automatic programming has focused on the roles played by some specific methods applied to some particular domains (Smith 1985; Wile 1984).

Automatic programming through deductive synthesis (Biermann and Gurho 1984) is based on the observation that constructive proofs are equivalent to programs because each step of a constructive proof can be interpreted as a step of a computation. If a constructive proof step is conducted by taking a part of a program and replacing it with a new (transformed) part, we have transformation methods (Bauer89, Burstall and Darlington 1977). In such a method, a program is derived from a formal specification by manageable, controlled transformation steps which guarantee that the final product meets the initial specification.

Two types of transformation rules are generally used. One is the general-purpose transformation rule which represents basic programming principles. The other is the special-purpose transformation which represents knowledge about the program's subject domain of contain types, such as streams (Barstow and Rood 1985), or lists (Dershowitz 1985).

In this paper, a method of deductive synthesis through transformation and an experimental system based on the method are presented. Four general-purpose transformation rules, based on Manna and Waldinger's previous work (Manna and Waldinger 1979), are proposed. They are Condition Formation, Recursion Formation, Generalization Formation, and Subsidiary Procedures Formation. These rules have been used in the synthesis system SSS (Wu 1985), a system for constructing array-manipulating programs from a given specification. In SSS, an specification language is formally defined which is similar to a natural language. SSS produces a recursive program which can be further transformed to an efficient target program.

In Section 2, basic programming principles are introduced together with four general-purpose transformation rules. In Section 3 the synthesis system SSS which uses the proposed method to synthesis array-manipulating problems is discussed. Finally we present some conclusions in Section 4.

2. FOUR GENERAL-PURPOSE TRANSFORMATION RULES

Automatic programming through transformation involves a sequence of transformation steps which transform a specification into an executable target program (see Figure 1)

\[ G_0 \rightarrow G_1 \rightarrow G_2 \rightarrow \ldots \rightarrow G_n \]

Figure 1: A sequence of transformations

In Figure 1, \( G_0 \) is the specification, \( t_i \) are transformation steps, and \( G_n \) is the executable target program. Usually, the \( G_i \) are called goals. Typically each transformation is correctness preserving.

In general a transformation step is performed by applying a transformation rule \( R_i \) to a goal \( OG \), where:

\[ R_i: \quad \underline{OG} \quad \frac{\rightarrow}{\downarrow} \quad C_i \]

which means that when condition \( C_i \) is true, goal \( OG \) can be replaced by goal \( TG \).

Example 1. If \( x, y, z \) are variables, and \( P \) is a predicate, then we have the following transformation rule:
This means that goal \( P(x, y, z) \) can be replaced by goal \( x, y, z := y, z, x \) (which is executable) if condition \( P(x, y, z) \) is true.

In general, to apply a transformation rule in a synthesis system, a theorem prover is needed to prove the condition associated with the rule. In order to construct a synthesis system with no theorem prover, a special type of transformation rules is introduced. A concept called complete transformation set is introduced in this paper. A set of rules \( \{ R_1, R_2, ... , R_n \} \) is complete with respect to OG if \( C_i \land C_j = \text{false} (i \neq j, i, j = 1, 2, ... , n) \) and \( \forall C_i = \text{true} \). The selection of transformation rules is discussed in the following four basic programming principles.

**Condition Formation:**

If \( \{ R_1, R_2, ... , R_n \} \) is a complete set with respect to OG, where \( R_1 \) is:

\[
\begin{array}{c}
\text{OG} \\
\rightarrow \\
\text{TG} \\
\rightarrow \\
C_i \\
\end{array}
\]

then the goal OG can be replaced by:

- if \( C_1 \) then TG1
- if \( C_2 \) then TG2
- ...
- if \( C_n \) then TGn

In a special case, if \( C_i = \neg C_{i1} \land \neg C_{i2} \land ... \land \neg C_{i1} i-1 \land C_{ii} (i=1, 2, ... , n-1) \) and \( C_n = \neg C_{11} \land \neg C_{22} \land ... \land \neg C_{n-1} n-1 \), then the goal OG can be replaced by:

- if \( C_{i1} \) then TG1
- else if \( C_{22} \) then TG2
- ...
- if \( C_{n-1} n-1 \) then TGn-1
- then TGn

**Recursion Formation:**

In the process of automatic programming if we encounter a subgoal that is an instance of the original goal, we can attempt to achieve this subgoal by forming a recursive call. To ensure that the introduction of this recursive call is legitimate, we must verify the input and termination conditions. We have proved (Wu 1985) that both of these two conditions are satisfied in the process of automatic programming in the SSS system.

**Generalization Formation:**

During the process of automatic programming if a subgoal is an instance of the original goal but of a more general goal, construct a new procedure to achieve the subgoal by a call to another new procedure to complete the more general goal, and to achieve the original goal by a call to that general goal. In SSS the generalization is produced automatically.

**Subsidiary Procedures Formation:**

Case 1: If some subgoal TG1 cannot be simply changed into the target language, then a subsidiary procedure is introduced to realize TG1.

Case 2: In using the Condition Formation, if condition \( C_i \) is a non-primitive logical statement, then \( C_i \) is substituted by procedure \( P(C_i) \) and \( P(C_j) \) becomes the new goal to be achieved.

3. **SYNTHESIS SYSTEM SSS**

This four general-purpose transformation rules discussed in Section 2 have been implemented in SSS. This system accepts a specification written in a high level language SL and produces an executable target program. SL is a language similar to natural language but is formally defined both in syntax and semantics. SL can express five types of problems: FIND ALL, TEST ALL, TEST SOME, FIND SOME, FIND THE NUMBER OF. The target program is a Pascal-like recursive program, which in turn can be transformed into an iterative program to increase the efficiency of the target program.

**Example 2.** In an array T, find all elements which have the same values as their index. In SL this problem can be expressed as: Find all \( T[i] \) where the \( i \)'s from \( A \) to \( B \) satisfy \( T[i] = i \). If this goal is denoted as \( G_1 \), SSS will find a (instantiated) complete transformation set with respect to \( G_1 \). This set is:

\[
\begin{array}{c}
(1) \\
G_1 \rightarrow A > B \\
Nil \\
G_1 \\
(2) \\
T[A] U G_2 \\
A \leq B \land T[A] = A \\
(3) \\
G_1 \rightarrow A \leq B \land T[A] \neq A \\
G_2 \\
\end{array}
\]

Where \( G_2 \) is Find all \( T[i] \) where the \( i \)'s from \( A + 1 \) to \( B \) satisfy \( T[i] = i \). In SSS, a Condition Formation rule is applied in this case. A procedure name will be provided either by the system or by the user. For this example, 'Find' is used as the procedure name.
Find (T, A, B) <= if A > B then Nil
  else if T[A] = A then T[A] U G₂
  else G₂

Since G₂ is a subgoal that is an instance of the original goal 1, Recursion Formation is used which achieves subgoal G₂ by forming a recursive call to G₁. We have now the following program:

Fine (T, A, B) <= if A > B then Nil
  else if T[A] = A then T[A] U Find (T, A+1, B)
  else Find (T, A+1, B)

Example 3: Find a maximal element in array T[...B]. The specification written in SR is: Find a \( T[2] \) where the i from A to \( B \) satisfies \( T[i] \geq \) all \( T[\ldots B] \).

Solution: Assume Max is used as procedure name, and T, A, B as parameters. Through Condition Formation we have:

Max (T, A, B) <= if A > B then Nil
  else if \( T[A] \geq \) all \( T[\ldots B] \) then T[A]
  else G₃

where G₃: Find a \( T[i] \) where the i from the \( A+1 \) to \( B \) satisfies \( T[i] \geq \) all \( T[\ldots B] \), which is not an instance of the original goal, but of a more general goal: Find a \( T[i] \) where the i from A to \( B \) satisfies \( T[i] \geq \) all \( T[C \ldots D] \). Assume GMax is used as the procedure name for this general goal, then:

Max (T, A, B) <= GMax(T, A, B, A, B)
GMax (T, A, B, C, D) <= ...

Again through Condition Formation, we have:

GMax (T, A, B, C, D) <= if A > B then Nil
  else if \( T[A] \geq \) all \( T[C \ldots D] \) then T[A]
  else G₄

where G₄: Find a \( T[i] \) where the i from A+1 to \( B \) satisfies \( T[i] \geq \) all \( T[C \ldots D] \) can be achieved by GMax (T, A+1, B, C, D) through application of Recursion Formation. By applying Subsidiary Procedure Formation, a new procedure Get (T, A, C, D) is introduced for \( T[A] \geq \) all \( T[C \ldots D] \), since this condition is a non-primitive logical statement. Finally we have:

GMax(T, A, B) <= GMax(T, A, B, A, B)
GMax (T, A, B, C, D) <= if A > B then Nil
  else if Get (T, A, C, D) then T[A]
  else GMax(T,A+1,B,C, D)

Get(T, A, C, D) <= if C > D then True
  else if \( T[A] \geq T[C] \) then Get (T, A, C+1, D)
  else False

In general, these executable Pascal-like programs are recursive. A method which remove recursion (Wu 1983) can be used to transform the programs into efficient, iterative target programs.

4. CONCLUSIONS

We have discussed an automatic programming through transformation methods which uses four general-purpose rules. This method has been applied to the synthesis system SSS, a system for constructing array-manipulating programs from a given specification. Five types of array manipulation are possible in SSS. SSS produces a recursive Pascal program which can be further transformed to an efficient iterative target program.

REFERENCES


Automatic Generating of Interesting Hypothesis
Concerning Gynecological Medical Data

Ladislav J. KOHOUT
The Center for Expert Systems and Robotics
Department of Computer Science
Florida State University
Tallahassee, Florida 32306, USA.

and
Isabel STABILE
Center for Environmental Toxicology
University of Florida
Gainesville, Florida

1 THE SCOPE OF THE PAPER

The paper describes how a method for automatic generating of interesting hypotheses from experimental data (GUHA) can be applied in a medical context, namely to the investigations of the relevant hypothesis describing the logical conditions characterizing the diagnoses of ectopic pregnancies. In order to use the algorithms that are available, the medical and logical meaning of what is interesting and relevant must be adequately defined. After briefly surveying the nature of GUHA algorithms we discuss in detail how it is applied to gynecological medical data, also outlining its advantages over the more traditional methods.

2 BASIC NOTIONS OF GUHA

GUHA (General Unary Hypothesis Automaton) represents a class of methods for generating interesting research hypotheses from experimental data by means of a computer. Interestness of a hypothesis is determined by the properties relevant to a particular application. The original GUHA method, introduced by Hájek, Havel and Chytíl in Prague was based on elementary logic theory. These GUHA procedures were devised to help the researchers in the choice of hypotheses, taking as the foundation of the generative principles Carnap's idea of induction models. Since 1968, when the first paper of Hájek et al. was published, the topic has developed into a fairly extensive research area, involving also methods of statistical and many-valued logic based inference.

A system of data-objects consisting of a finite collection of objects is examined to see whether or not the objects possess various properties. These observations are recorded in a relation represented as a matrix of the size $mn$, where $m$ is the number of the objects entering in to the data-model and $n$ is the number of properties. Each row, say $i$, contains the logical values of the properties associated with the object $o_i$. Hence the information concerning the given objects can be described by the observational structure

$$\mathcal{M} = (< m, p_1, p_2, \ldots, p_n >$$

where $m$ is a nonempty finite set of objects and each $p_i$ is a mapping from $m$ into some given set of logical values, depending on the version of the GUHA method used. The logical values can be just true and false (the case of classical logic), probabilities or values from the interval $[0, 1]$, representing the valuation space of a many-valued logic.

The hypotheses generated about a particular set of objects are logical formulas defined over the set of properties characterizing the objects. GUHA method is then defined as a set of algorithms that will generate interesting hypotheses about a given set of objects. When the method is applied to a particular scientific domain, the abstract objects will represent the physical objects of that particular domain and the properties are the features identified experimentally by the scientist/observer/experimenter [1],[2],[9],[4].
Because generating all the formulas up to the length \( l \) would produce too much information about the logical structure of the problem, only part of which is directly relevant to it, the formulas have to be generated selectively. Two kinds of restrictions on the type of formulas to be generated are applied:

1. logical restrictions;
2. pragmatic restrictions.

Logical restrictions allow us to generate only such formulas that are necessary for expressing all the relevant information. The pragmatic restrictions eliminate those formulas that are for some reason not interesting to the scientist/technologist user of the the GUHA. Pragmatic restrictions, of course, must have their logical form, but that is generated from the semantics of the application domain and translated into its logical form. The logical restrictions in (1) above, on the other hand, are generated from purely logical considerations independent of the domain of application.

3 INCOMPLETENESS OF DATA IN MEDICAL APPLICATIONS

GUHA algorithms based on two-valued classical logic assume that for every property in the observational data it is known whether an object possesses it, or not. Three-valued extension of GUHA can deal with incomplete information. It interprets the logical value 1 as a property being present, the value 0 as it being absent, and 0.5 as the lack of information concerning its presence or absence. It was proved by Rauch [6] that Klenke-Dienes three-valued logic connectives give the correct logic to deal with incompleteness of information in data-models in cases in which part or all missing information may be later supplied.

In a clinical situation, medical data pertaining to a patient are very often incomplete. For this reason two-valued GUHA method is not always applicable in every case, and a version based on a three-valued logic is needed. A three-valued logic based GUHA system using a new search algorithm devised by Sheebeeb [4] has been implemented (Kohout and Sheebeeb [6]) in PASCAL and runs on SUN workstation under the operating system UNIX.

4 INVESTIGATION OF THE LOGICAL STRUCTURE OF MEDICAL DATA CONCERNING THE DIAGNOSIS OF ECTOPIC PREGNANCY

4.1 An Overview of the Medical Problem

The present paper describes the use of this system for analysis of medical data concerning ectopic pregnancy as well as the ways in which new hypotheses concerning this condition are generated. To determine how to logically describe what are the interesting hypotheses a detailed knowledge of the medical conditions is required. Thus close cooperation between an expert physician/gynecologist and a logician is required. From the description that follows, it will become obvious that this medical condition is important albeit difficult to fully evaluate by conventional methods.

An ectopic pregnancy occurs when the fertilized ovum develops and grows outside the uterus. Women with this condition may present with a variety of symptoms e.g., amenorrhea, abdominal pain and vaginal bleeding (Stabile et al. [7]). Unfortunately this combination of features is common to other gynecological conditions. Therefore, if the clinician cannot rely on clinical features to make and exclude the diagnosis, additional tests become necessary. The choice of which particular test to use in the clinically stable women i.e., ultrasound scanning or biochemical tests, and at which cut off level, is often difficult and is dependent on the facilities available in the given institution (Stabile et al. [8]).

Our recent observations on the value of ultrasound examination in a group of women with subacute symptoms suspected of harbouring an ectopic pregnancy, have highlighted the importance of careful ultrasonic examination of the pelvis including a number of ultrasonic features such as uterine area, endometrial thickness, and adnexal volume, whenever this diagnosis is considered (Stabile et al. [9]). However, the sole use of ultrasound has not attained the sensitivity and specificity of biochemical tests such as human chorionic gonadotrophin measurements. The aim of this work is to evaluate different cut off levels of various biochemical and ultrasonic parameters in varying combinations in order to improve their diagnostic efficacy in this life threatening condition.
4.2 Choice of the Semantic Descriptors of the Problem

The general model (cf. Sec. 2 above) consists of objects and properties. To provide the clinical semantic interpretation of the relations involved, we have to select the appropriate clinical concepts as the names of the unary predicates designating the properties of the experimental data describing the ectopic pregnancy condition.

Using the above described model with its clinical interpretation determined by the semantic descriptors which forms a part of the theory, we then generate the interesting hypotheses contained in the test (i.e. observational) data. The logical restrictions (cf. Sec. 2) are given by the choice of the type of GUHA method. The pragmatic restrictions are determined by the types of medical questions we have to formulate, in order to generate and subsequently interpret the hypotheses (logical formulas) generated by the application of the GUHA algorithm.

The following is the sample of meaningful questions that are interesting from the clinical point of view:

- Does any single parameter logically imply a combination of up to k parameters?
- Does any combination of m parameters (i < m < j) logically imply a single parameter?
- If one parameter is negative, what else is negative?
- If a selected group of parameters is positive, what else is positive?

These type of questions are being asked of the data (i.e. an observational model in GUHA terms) composed of the patients with positive ectopic pregnancy findings.

There are a number of negative groups about which analogical questions can be formulated, within the context of the negative results. This is can be done either on:

- Independently on each negative group.
- On the group composed of all the negative groups.
- On the group containing the patients with some other abnormal findings that appeared in the clinical test trial.

Whereas complete test data may be available for some biochemical data [such as human chorionic gonadotropin (hCG), Schwangerschafts protein 1 (SP1) and Pregnancy associated plasma protein A (PAPP-A)], the same may not be the case for all other biochemical test parameters (e.g., alphafetoprotein (AFP), or pregnancy protein 12 (PP12) on all patients suspected of having an ectopic pregnancy. In this situation the GUHA method allows the scientist/physician to identify interesting hypotheses, even though the data set is incomplete. This is important with respect to the calculation of the accuracy of a diagnostic test. Accuracy is usually described in terms of sensitivity, specificity and predictive values (positive and negative) as defined below.

Sensitivity is defined as the proportion of persons with a condition who correctly test positive when screened i.e., the ability of the test to detect the condition. Specificity is defined as the proportion of persons without the condition who correctly test negative when screened i.e., the ability of the test to exclude the condition. The positive predictive value is defined as the proportion of positive results that are actually correct (true positives). This is partly dependent on the prevalence of the disease in the population being studied. The negative predictive value is defined as the proportion of negative results that are actually correct (true negatives).

Therefore, although incompleteness in data sometimes results, the utilization of reliable information on these sets of incomplete data will allow the clinician to retain those tests that have the desired accuracy, as defined above.

An example of using combinations of tests when complete data are available is displayed in the figure below. This refers to the predictive value of ultrasound findings taken singly and in combination in women in whom the diagnosis of ectopic pregnancy was confirmed surgically (Stabile et al. [9]). In this example, the interesting hypotheses generated are the disjunctions (simultaneously positive combinations) of the tests. The three valued GUHA algorithm makes it possible to extend investigation of the various combinations of these tests to incomplete sets of data, be they clinical information, biochemical data or sonographic findings. It is precisely this feature which gives the GUHA method an advantage over the currently available methods of data analysis.
References

1. Hájek, P., Havel, I., and Chytíl, M.

2. Hájek, P., Havel, I., and Chytíl, M.
The GUHA method of automatic hypotheses formation.

3. Hájek, P. and Havránek, T.

4. Sheebeeb, N.D.
*The Generation of Interesting Hypotheses from Incomplete Data by Means of GUHA Method.*

5. Rauch, J.
Ein Beitrag zu der GUHA Methode in der dreiwertigen Logik.

6. Kohout, L.J. and Sheebeeb, N.D.
Design of virtual architectures for GUHA inferential systems.
in press.

7. Stabile, I., J.G, Westergaard, and Gruzdinskas, J.D.
Ectopic pregnancy: diagnostic aspects.

8. Stabile, I. and Gruzdinskas, J.G.
in press.

9. Stabile, I., Campbell, S., and Gruzdinskas, J.G.
Can ultrasound reliably diagnose ectopic pregnancy?

Fig. 1. Predictive value of ultrasound findings taken singly and in combination. \(\times\), Any abnormality; \(\bigcirc\), free fluid in the pouch of Douglas; \(\bigcirc\), uterine area <20 cm\(^2\); \(\triangle\), adnexal mass volume >10 ml; \(\bullet\), endometrial thickness >10 mm.
1 ABSTRACT

NASA Space Shuttle liquid hydrogen (LH2) loading is a complex and critical operation. This paper presents the results of a continuing research effort focused on revealing how this process may be made safer and more reliable. Upon investigation it was found that model-based reasoning offers the most useful approach to the design and construction of an appropriate diagnostic knowledge-based system. The theoretical basis for implementation of model-based reasoning diagnostic system is presented. An extension to existing theory is offered that expands the method’s application to fluid transfer problems. A model-based reasoning diagnostic system is being implemented for the detection and resolution of failures in the Space Shuttle propellant loading system during launch operations.

2 INTRODUCTION

The LH2 propellant loading of the Shuttle during the hours before liftoff is a critical and complex operation. The loading is performed late in the countdown because at -423°F the LH2 quickly boils off from the Shuttle and has to be replaced. The transfer of the LH2 from the facility storage tank to the External Tank (ET) of the Space Shuttle involves over 20 complex operational phases comprising system chilldown, slowfill, fastfill, topping, replenish, and terminal count. Therefore, many sensors are required to ascertain the state and progress of the fluid transfer operation. Problems in any phase of this process could result in missing the launch schedule. For this reason the aforementioned sensors are used to monitor the loading process for failures and, in principle, to facilitate a timely diagnosis.

However, a difficulty exists in that the particular failure causing a problem is not always evident from the sensor data. In fact, it is often the sensor which has failed. The present system does not always provide a reliable and timely diagnosis of system and sensor failures. Due to lags in the detection and diagnosis of loading problems -- delays have been experienced and safety could be compromised. A quicker and more dependable method of detecting and automatically diagnosing failures is required.

Presently sensor data is collected and compared to real-line and historical limits by monitoring software. When a sensor reading exceeds a specified limit, the system flags the attention of the monitoring engineers who subsequently perform a manual diagnosis. For a potentially serious failure, the propellant loading control software puts the system into a safe mode and flags the operator. Under this system, a problem may not be noticed until it has already become serious (i.e., a sensor reading has exceeded its limit). More often, a problem is automatically detected and the loading system put into a safe mode only for the engineers to determine many minutes later that it was just a sensor failure. The current situation is, therefore, undesirable because of the risk to both safety and the launch schedule.

The purpose of this paper is to review the present situation, consider alternative procedures, and recommend a specific solution by which the loading process may proceed more safely and reliably. The selected software system should be able to monitor, diagnose when discrepant, and report the propellant loading system state in real-time despite sensor failures.

3 REQUIREMENTS

The Space Shuttle Liquid Hydrogen Loading System (Figure 1) consists of a network of fluid lines and valves connecting the facility storage tank to the External Tank (ET) of the Space Shuttle. Each of the two tanks, the ET and the facility storage tank, has a vent valve, a flare stack to burn the vented GH2, a drain/fill line, and a pressurization system. There are several pressure sensors and each valve has at least one position sensor. The onboard ET sensors indicate the position of each vent valve and drain valve as well as the tank pressures, liquid levels, and pressurization rate.

Proceedings of the 3rd Florida Artificial Intelligence Research Symposium, 1996, by the Florida AI Research Society

c. 1990 FLAIRS 0-9620-1732-9/90/0350-0214
Problems occur during complex fluid transfer operations, such as the Space Shuttle Propellant Loading. Sometimes a component will fail, such as a valve getting stuck, but more often it is one of the numerous sensors which fails. Critical system components are redundant so that valve failures can usually be bypassed. Some of the more important sensors are also redundant. When a redundant sensor fails the presence of a failure is evident and another sensor still provides the correct reading. Quick recognition of the failure greatly enhances system safety and reliability.

There are many system components which do not have redundant sensors. Sensors become very expensive when the total cost of calibration, installation, and data channels is considered. When solitary sensors fail it is difficult to tell immediately whether the sensor or the component failed. Even when a measured value has two sensors reading differently, indicating that one has failed, it may not be immediately evident which sensor is providing the correct reading.

An anomalous sensor value could be caused by a fault in the measured component, a related component, or even the sensor itself. Failed sensors are especially pernicious because they can cause a shutdown when the system is actually working properly. In fact, a sensor failure caused a Space Shuttle main engine premature shutdown on a Shuttle Spacelab flight. Only by quickly calculating the consequences of such an indicated failure and checking for these consequences at other sensors can an unnecessarily conservative response be avoided.

An automated system which diagnoses complex systems in seconds rather than minutes or hours could avoid unnecessary system shutdowns and facilitate safe system operation. When Shuttle engineers are struggling to meet a launch window this diagnosis and restart time can be critical. From operational experience it is especially important that the envisioned system be able to distinguish sensor failures from system component failures.

4 THE APPROACH

4.1 Improved Hardware vs. Diagnostic Software

Diagnosis is currently performed by human operators who continuously monitor the loading system operation. When any of the sensors do not read within prescribed limits, the controlling software recognizes that some fault has occurred and cycles the loading system to a safe mode. The monitoring engineers diagnose the fault from the sensor data and schematics and determine how to restart the system. This process can take thirty minutes, as the schematics for an operational system are typically voluminous. Finding the relevant system configuration data and making the necessary calculations hinders the engineers from further system monitoring. Even after the nature of the problem is resolved, planning and implementing the restart can cause a significant delay as well.

While obtaining more reliable components will decrease the failure rate, it will not solve the problem. There are sufficient components and sensors in the loading system that failures are inevitable. These failures do not necessarily lead to a debilitating situation as there is sufficient hardware redundancy to bypass most failures. Unfortunately, this redundant hardware also makes the system more complex, thus complicating diagnosis. A diagnostic software system can therefore, best improve system safety and reliability by making better use of the existing hardware rather than by obtaining additional hardware.

4.2 Diagnostic Software: Conventional Software vs. Expert System

Automated diagnosis of a complex system is generally not practical with conventional software because of the tremendous number of system states and failure modes. For LH2 loading, explicitly coding every possible mode involves at least 20 valves either open or closed giving 2^20, or over 1 million, possible valve position combinations. Even allowing for only single valve failures, there are the 33 normal valve configurations each with 20 different single-valve failures yielding 660 single-failure valve-configuration combinations. Factoring sensor failures into the number of possible states makes it clearly desirable to have the failure modes be implicitly part of the system definition. The model-based reasoner can derive any system state from the system functional definition, thus alleviating the need to explicitly enumerate all possible operating and failure modes.

Expert systems are well suited to system diagnosis. The system data and relations can be declared separately from the diagnostic logic. With an inference engine which performs the diagnosis, the system data and relations are not complicated by intertwined diagnostic logic. Moreover, in this application the diagnostic logic is complicated by the sensor data being no more reliable than the system components. The diagnostic logic must account not only for component failures but also for failure of the sensors through which it observes the system status. Because a complex operational system can be encoded separately from the complex diagnostic logic, an expert system is better suited to this problem than conventional software.
4.3 Expert System: Case-Based Reasoning vs. Model-Based Reasoning

There are two principal types of expert systems: case-based reasoning (CBR) and model-based reasoning (MBR). Most Expert Systems use CBR. CBR encodes an expert's empirical knowledge of a system's behavior into production rules. MBR exploits the structure of a system and the behavior of its components to reason about system behavior.

CBR has been applied to many domains, chief among them medical diagnosis. Mycin (diagnosis of infections) and puf (diagnosis of pulmonary disease) are based upon the experience of medical practitioners in their respective domains of application. These expert systems use the inference engine to resolve the logic defined in the rule base. Internist (Internal medicine diagnosis) gains increased generality and power by separating the diagnostic logic from the empirically known disease-symptom relations. CBR is well-suited to medical diagnosis since medical diagnosis is fundamentally based on empirical relations.

More structured domains give rise to the possibility of expert systems which represent knowledge by structured objects. R1 (configuror of VAXes) and macsyma (solver of complex mathematical problems) derive their power and generality from this structured representation of their domain knowledge.

MBR is a relatively recent and not as yet widely used foundation for the construction of expert systems. MBR is based on the use of a system model separate from the logic which reasons about the model. MBR has been applied to troubleshooting digital electronics and electromechanical systems. In these structured domains a functional model of the system already exists from the design definition, thereby averting the need for the empirical description required by CBR.

In the case of the Space Shuttle LH2 Loading system, the availability of a functional model makes MBR a clear choice over CBR. By employing the model-based rather than the case-based reasoning paradigm, the knowledge acquisition bottleneck is largely avoided. The model is more cheaply acquired and far more reliable and complete than an empirical description can be. The MBR model is more complete because it provides an implicit enumeration of all the possible system states. The number of possible system states is quite large in most operational systems because the number of states increases exponentially with the number of independent system parameters (in this case, valves). By virtue of this complete implicit characterization of system behavior, model-based reasoning is reliable -- even in novel situations. The functional model basis makes MBR systems more reliable and complete than CBR systems.

4.4 Previous Work in Model-Based Reasoning Systems

MBR systems are closely associated with the diagnosis of devices. MBR has principally been applied to systems based on combinational logic. Extensive theoretical work has been done applying model-based reasoning to the diagnosis of digital circuits (Davis 1984), (DeKleer and Williams 1987). The classic example in these papers is an adder/multiplier circuit. From the algebraic inputs and outputs of the integer arithmetic circuit, a fault can be localized to one of the individual adders or multipliers.

More closely related previous work has successfully diagnosed complex engineered-systems in real-time using their functional descriptions. NASA has used LES (LOX Expert System) to diagnose faults in the Space Shuttle Liquid Oxygen Loading System during countdowns (Scar1, Jamieson and Delanue, 1987a), (Scar1, Jamieson and Delanue. 1987b). LES diagnoses the equipment which controls the flow of LO2. LES seeks to determine whether the control equipment (valves, pumps, hydraulics) is configured as commanded. Any faulty control component (relay, fuse, circuit) which causes the sensor output to be inconsistent with the command inputs is identified and displayed (New 1985).

LES diagnoses by using the software commands as inputs for a simulation of the control equipment. The simulated sensor outputs are compared to the corresponding observed sensor values. Where the outputs are not in agreement LES isolates the source of the discrepancy by inferring and comparing system values to find a fault which is consistent with the sensor data.

LES pioneered using MBR for sensor failures and in electromechanical systems (circuit relays, hydraulics, and valve movement). To diagnose a fluid-mechanical system, MBR must be extended from the local combinatorial logic of arithmetic and control circuits to the systemic relations of fluid transfer. Fluid transfer is systemic because the pressure and flow rate at any given location are a function not only of the local valves but are dependent upon the configuration of the overall fluid network. Because of these differences between the LES capabilities and the fluid transfer requirements, as well as language and machine compatibility problems, LES is used only as a model for the development of the LH2 loading diagnostic system.

5 AN APPLICATION OF MODEL-BASED REASONING TO LH2 LOADING

This section presents the mathematical model of the LH2 loading system, the required capabilities of the diagnostic system, and the theoretical basis for this diagnostic system. The present math model is quasi-steady state, applicable only during steady-state conditions, so that the brief system transients between flow phases are neglected at a great savings in initial
development complexity. The capabilities of the diagnostic expert system were defined by the engineers who are presently responsible for monitoring the LH2 loading. To be able to diagnose a fluid transfer system subject to real-time operational requirements, the theoretical basis of the diagnostic system required an extension to MBR.

5.1 Defining the Model

MBR requires a functional model of the system being diagnosed. The mathematical model should include the sensor readings, the significant system parameters, as well as parameters measuring the functionality of the components which can fail (e.g., valve position).

The tanks and fluid lines are presented in Figure 2. The equation relating the flow rate and pressures in the fluid lines is simply Darcy's equation, \( \text{Mdot} = K \cdot \sqrt{P_1 - P_2} \), where \( \text{Mdot} \) is mass flow rate, \( K \) is the flow constant, and \( P \) is the pressure.

![Figure 2: LH2 Loading System Schematic](image)

The pressure drop across a segment depends on the flow rate through it. By continuity, the flow rate through a segment depends upon the flow through the adjoining segments. The system behavior is determined by calculating an equivalent fluid resistance for the parallel fluid lines and summing the series fluid lines for a single flow resistance. From this system flow rate, individual flow rates may be derived.

Valves have from 1 to 4 sensors. Pressure is measured in the storage tank, at two places along the fluid transfer line, and in the ET with redundant pressure sensors. The equation relating a sensor to its measured value is, of course, identity.

For diagnosis, an engineer prefers to view a system using a schematic representation of its components. The schematic depicts the organization of the values and formulas (which relate the values). Likewise, a diagnostic program will work best using a representation of the system which is isomorphic to a schematic; the data structure contains the structure and organization of the system but in a computer usable form. The calculation of system values implicitly requires additional networking for the system schematic.

Presently, a computer data acquisition system reports real-time sensor data to monitoring engineers. The diagnostic Expert System can therefore receive a state vector from the existing system. For a quasi-steady state math model, the diagnosis is, of course, only meaningful when the fluid transfer system is in a quasi-steady state: so, only after a valve movement is completed can the diagnostic system evaluate the system state. These diagnoses must be performed concurrent with loading system operations in real-time. The diagnostic logic is simplified by the system initially having to diagnose only the more probable types of failures. Operationally, the components which fail are the valves and sensors.

The system output should be both quickly and easily comprehended. The quantitative calculations which were made should be available, to support the reasoning of the diagnosis. To facilitate quick understanding of the system state, a few qualitative classifications are the best initial report. Overall, is the system in a fill, drain, or hold mode? Are the ullage pressures within acceptable limits, and are they rising or falling? Have any sensors or components failed?

For the qualitative description to be credible it must be supported by quantitative values. Stating that the flow mode is fill, drain, or hold must be supported by the actual flow rate. For a violation, both the actual system value and the system requirement should be displayed for comparison. With the presentation of the values and their derivations the assertions about the system state can be supported.

5.2 Defining the Reasoning

The diagnostic process can be divided into four parts: fault detection, suspect generation, discrepancy resolution, and system state reporting. Fault detection discovers the inconsistency in the sensor readings. Suspect generation determines the set of faults which could have caused the inconsistency. Discrepancy resolution examines the suspects to determine which is the cause. System state reporting presents the results of the diagnosis.

5.2.1 Fault Detection. Faults cannot necessarily be isolated directly. An anomalous sensor reading could mean either a faulty component or a faulty sensor. In this loading system, because all the valves have sensors, all the valve positions, correct or failed, are detected directly by the sensors. Sensor failures, however, can present a distorted view. A sensor failure could a priori appear to be a valve failure. However, after inferring the related system values, that particular sensor measurement could be determined to be inconsistent with the overall state of the system.

Whether the failure is a component or sensor, it causes a local inconsistency in the model. The model asserts a functional relationship which is no longer valid. This local inconsistency can be detected by applying the affected part of the model in a calculation which can be compared to the observed system. In previous work, LES and the multiplier/adder circuit,
consistency checking is accomplished by checking for discrepant outputs. Because the loading monitoring system is based strictly on sensor data, inconsistencies must be detected without the command inputs.

All sensor readings can be verified by calculating the consistent local flow rate. The flow rates along a serial flow path must be, by continuity, equal. When the sensor readings are not consistent, there is a fault. For example, if a valve position sensor is faulty then the calculation of just that one flow segment is inconsistent. If it is a pressure sensor that is faulty then the two adjoining segments will be inconsistent both with each other and with the rest of the system.

In general, sensor failures can be differentiated from component failures by exploiting the functional relations which lead from that one sensor to other known values. The state of a component, by definition, affects the state of the system in which it is configured; otherwise, it is not truly a part of the system. So the state of the component can be inferred from the state of the system which it affects. For instance, the pressures upstream and downstream of a valve may be consistent with either an open valve or a closed valve. From this calculation the discrepancy can be isolated to either the sensor or the component, thereby determining whether system operation can proceed normally or not.

There must be a minimum number of sensors even to detect an inconsistency. For a system represented by an equation, or set of equations, in three variables there must be at least two known values to determine system state, 3 to observe a discrepancy, and 4 to resolve an observed discrepancy. In general, for a system with N degrees of freedom and S sensors:
- If S-N: an inconsistency is not detected;
- If S=N: an inconsistency can be detected; and
- If S>N: S-N inconsistencies can be detected and resolved.

This, of course, neglects certain transitivity problems. However, in general, the more sensor data available, the more inconsistencies can be detected and resolved.

Many failures can be the cause of a detected inconsistency. The failure modes of concern to this system are limited, fortunately. The principal component failure mode is a valve in the wrong position due either to the valve being stuck, its control equipment having failed, or as the result of a bad command. More common is a sensor failure which is due to the sensor being stuck or dead. Also of concern is the mass of propellant in the ET, which changes slowly enough to be estimated accurately, and the ullage pressures in the two tanks, which are measured with extra redundancy.

5.2.2 Suspect Generation. When the sensors do not clearly indicate the failed component a list of possible suspects must be generated. Only reasonable faults should be evaluated and the suspect space should be incrementally created and examined.

For a discrepant value the suspect list is the calculations which were used to derive the value and the values which were used in the calculations. The suspect generation is recursive because the calculations used to derive the values used are themselves suspect. But the suspect list is bounded when a value is reached which has been independently confirmed as correct.

It is reasonable to assume that only one failure has occurred at a time. The single failure hypothesis requires that the time between sensor snapshots is substantially less than the mean time between failures. In the case of Space Shuttle LH2 loading, the mean time between failures is hours and the sensor sample rate is seconds. Hence, the single failure hypothesis can be used.

In most cases valve sensors will identify the initial suspect valve. Otherwise, discrepant pressure readings indicate either a valve position or tank pressure error. The pressures and pressure gradients in the observed system, while easy to observe, are hard to use for diagnosis because they depend on more than just the local valve(s). Of course, when the pressures read too high on one side of a valve and too low on the other side, it is probably that valve which is out of configuration. To make all the computational paths explicit, the physical schematic is amended to show all the local and systemic calculations.

Inconsistent readings in the tanks can be used to reflect the state of the vents, drains, and pressurization system. The range of faults that can be diagnosed in the tanks is limited to values which are in quasi-steady state, such as fill rate or pressure decay rate. Ullage pressures are directly sensed and can be calculated but discrepant ullage pressures can be hard to explain because they are dependent on the past as well as present configuration.

5.2.3 Discrepancy Resolution. Discrepancy resolution requires deciding between conflicting values. The decision is based upon superior credibility. With three sensors, the two consistent sensors outweigh the one apparently discrepant sensor. With two sensors, the tie vote must be resolved with a calculated value. With one sensor, two independent votes must be obtained from system calculations. The sensor discrepancies are resolved by comparison to a calculated value. By the single failure hypothesis the rest of the system values may be assumed correct. An expected value for the sensor location is calculated which discriminates between the two sensors. The faulty sensor is marked and the resolved value is used to determine the system state.

Closed-form calculations are employed with only a minor computational cost. Closed-form calculations can determine pressures and valve positions easily -- as long as the flow path is simply between two tanks. While more complex flow paths can also be solved with closed form calculations, it is in general simpler to implement a general solution using an iterative simultaneous equation solver, although at much greater computational expense.
Concern with avoiding excessive run-time overhead prohibits the initial use of iterative network solvers. Diagnostic capability is therefore restricted to the case of not more than a two source/sink flow path in this application of two tanks and a fire stack.

5.2.4 System State Reporting. The overall system state can be determined by use of the resolved system values. The flow between the two tanks may be ascertained from the valve positions and tank pressures. From this flow value the system state is finally classified as either one of several fill modes, drain modes, or hold modes. Through a graphic schematic interface the pressures and flow rates throughout the system may be labeled and the valve positions may be depicted. Any anomalous values will be flagged to the monitoring engineers for consideration.

6 CONCLUSION

An expert system employing model-based reasoning is the best available solution to the task of creating an LH2 loading diagnostic system. It is preferred because it makes it easier to implement sound diagnostic logic and a complete and reliable system description than it would be with a different software tool. Also, because the diagnostic logic and the system description are separate, MBR promises to be simpler to implement and maintain. It was necessary to extend the current state of MBR capabilities to enable the envisioned system to perform consistency checking without system inputs and to diagnose systemic behavior. Consistency checking enables examining a system without using its actual inputs. Instead the outputs are correlated with one another by inferring the inputs and then calculating the expected outputs from these inferred inputs. The systemic behavior, the pressures, and flow rates in the fluid transfer system required new diagnostic techniques beyond that of the combinatorial logic upon which previous MBR systems have been established.

7 ACKNOWLEDGEMENTS

We would like to thank the many people who helped us. Lisa Emery, Lyle Ferguson, Larry Hartley, Kevin Kelleher, Steve Lim, Paul Kroniaghe, Mark McCandless, Jack Reiner, Ethan Scarl, Tom Schneider, and Bernard Widofsky made suggestions which improved the presentation. Sylvia Meyers and Margaret Peppersack provided valuable support finding references. This work was supported in part by NASA contract #NAS8-33708.

8 REFERENCES

Knowledge Engineering in LILOG

G. Klose, K. v. Luck

IBM Germany
Scientific Center
Institute for Knowledge Based Systems
P.O. Box 80 08 80
D-7000 Stuttgart 80

Abstract
When talking about Knowledge Representation, one usually has to differentiate between at least two views or readings summarised under this heading. The first one deals with methods of representing knowledge in the sense of considering logical properties of formalisms or associated inferential aspects. The second one, on the contrary, aims at an adequate coding of certain domains of knowledge by using - among others - logical formalisms as representational means. Knowledge Engineering reflects two central concerns when talking in terms of the second reading, i.e., ontological aspects of knowledge on the one hand and relevant decisions during the engineering process on the other. In what follows, we will first outline the domain under investigation. The background knowledge necessary for understanding the given texts will be sketched, with special attention given to some basic design decisions and categories of the model. Considerations of relevant criteria for validating the model will close this chapter.

1 The Domain

The necessity of adequate background knowledge for purposes in the field of intelligent simulation of text understanding constitutes a well known requirement within AI. If you do not want to build up a system for a specific task (e.g., a typical expert system) but for comprehending normal sentences understandable by normal people, you have to reconstruct a domain not only in a (relatively) task-independent way but also on a level of common sense knowledge (e.g. [Hayes, 1979] and [Hobbs and Moore, 1985]). In addition to considerations of computational efficiency, completeness, consistency and epistemological adequacy you have to deal with issues of transition between two forms of representation, i.e., natural language on the one hand and structured conceptual information on the other.

This undertaking meets a central concern for AI research, with no clear elaborated solutions so far. Therefore, several restrictions have to be made for rendering a reconstruction of world knowledge feasible. First of all, you have to define a specific domain, a restriction which determines the breadth of the model. Second, you have to evaluate the required depth of text understanding, and with this decision the depth of the model. These constraints can be illustrated by the following questions:

- What are the concepts you are interested in?
- And what is the knowledge about these concepts?

In what follows, we will focus on several scoping decisions for the relevant domain in LEU II. After that, the basic categories of

In what follows, we will focus on several scoping decisions for the relevant domain in LEU II. After that, the basic categories of this model will be presented and discussed, with special attention to the simultaneous use of shallow and deep knowledge for the targets in question. An outlook on some questions of compatibility with language surface structures will be given at the end.

1.1 The Scenario

The domain chosen for LEU II are textual informations about sightseeing tours in a larger German city (Düsseldorf). The task for LEU II is the understanding of these textual informations and - for proving the successful understanding - a limited question answering about topics mentioned in the texts.

In order to limit the scope of questions (in the sense of simulating a realistic, not a real dialog) the following story was constructed:

A group of people is joining a workshop for one week in Düsseldorf. One afternoon is scheduled for a walk in the city center. LEU II should be used for providing information about the city center of Düsseldorf.

1.2 The Sources of Information

To acquire the information necessary for this domain, several travel agencies and information centers were inquired about suitable informations for planning an afternoon. In addition, some travel guides were analyzed. For completing the information source, specific data were collected by telephone (e.g., missing opening hours) and by visiting the city center of Düsseldorf (e.g., price of food in some restaurants).

In a second phase, the incoming information was classified and the ultimate corpus of information about the city center was chosen. Three different kinds of textclasses were produced in order to reduce the feasibility of constructing special purpose procedures only suited for just one text.

1.3 The Texts

The three types of text chosen can be characterised as follows:

1. Some 20 short texts were produced describing single sightseeing items of Düsseldorf.
2. One text was a (moderately changed) description of a part of the city center taken from a travel guide.
3. One text was designed to be a (fictive) short story about spending such an afternoon.

For the classes of textual information, a couple of questions and answer pairs were chosen to describe the necessary depth of understanding minimally required for answering the questions posed in an adequate manner. One example of (1) and the beginning of (3) will be the basis for further discussions and therefore presented in connection with one question answer pair, respectively, in the following:

Die Kunstakademie, die 1773 aus einer Zeichenschule entstand, ist in einem aus der Gründerzeit stammenden Gebäude in der Eiskellerstrasse untergebracht. Es wurde 1879 nach den Plänen des Architekten Riffert errichtet. Das Gebäude, das durch seine breiten Vorhöfe auffällt, wurde 1945 restauriert. ...
The semantic content of this type of text can be characterized as more or less object oriented clusters of information with a certain number of relevant data like the kind of object, location and opening time.

Therefore, a typical question-answer pair looks like:

q: Wo befindet sich die Kunstakademie?  

a: In der Eiskellerstraße.  

The short story starts as follows:

in der ersten Maiwoche fand in Düsseldorf der große LILOG-Workshop statt. Nach der anstrebenden Projektsetzung am Dienstag machten die Teilnehmer des Workshops am Mittwoch einen Ausflug in die Düsseldorfer Innenstadt. Der Bus brachte sie von ihren Hotels am Rhein in rund 20 Minuten zum Hofgarten. Kaum waren sie angekommen, begann es zu regnen, trotzdem unternehmen sie einen etwa einstündigen Spaziergang durch den Park....

The chosen question focuses on the sequential event structure of the story mainly, information about single objects and their relations serving only as a kind of background knowledge.

q: Wann begann es zu regnen?  

a: Nachdem die Gruppe im Hofgarten angekommen war.

These different types of texts enforce us to represent the background knowledge in fairly different ways, taking into account the inferences necessary for obtaining adequate replies to the questions concerning the textual informations.

2 The Model

2.1 Task-oriented Decisions

2.1.1 Representation of Objects/Entities and Events

The types of questions cited above illustrate the necessity of representing the knowledge extracted from the texts following at least two different aspects. Typical proofs of an adequate processing of single item descriptions - as in the text types (1) and (2) - can be found in those answers reproducing the semantic content of the texts in a literal or paraphrased way. In these cases, the selection criterion for relevant information has to be the attribution to the entity or object focussed by the question. We assume the most general question of this type to be something like What do you know about... or Can you tell me something about... In our example we find a more specific request about the location of the sightseeing object.

When we look at the questions concerning text type (3), realistic text understanding remains obviously not object-oriented. Instead, there is a shift of focus towards the structure of events, including aspects of sequence, directness, resulting changes (e.g. of location) and the choice of entities involved.

Consequently, we have to build up the knowledge base according to (at least) two structuring principles. On the one hand, we need

special information (attached to object-like constructs) which can be retrieved by using the concerned entity as an access key, and on the other hand there has to be a kind of event concept, taking into consideration those aspects of meaning which can also be found in the object-oriented part of the ontology as well as transcending features like change, exclusion of two contradictory states at the same time (e.g. two subsequent locations of the same institution), sequential structuring (e.g. It starts raining after some other event) and so on.

2.1.3 Requirements of Temporal Variability

Another important kind of decisions - relevant for both types of information - concerns problem specificity. One example might be the kind of temporal information that should be provided in the focussed application domain. It should be noted here that two versions of temporal reasoning do not make sense in this setting: (a) temporal variability of the whole structure and content of the ontology, corresponding to a fundamental change in the represented view of the world, and (b) database solutions like updating, where new data replace older ones. The reason for our decision against the latter version is that we want to keep information like names of streets changing through the centuries.

The texts and the expected questions suggest an isolation of three different types of temporal reasoning. We find

1. dates like years or centuries associated with artifacts

Es wurde 1879... errichtet.  

2. processes and events following one another, but without any exact mapping to the calendar

Nach der anstrebenden Projektsetzung... machten die Teilnehmer... einen Ausflug...  

3. single aspects of objects changing over the time

Ab 1803 beinhaltet das Bauwerk Büros, und seit... 1968 wurde es als katholische Pfarrkirche genutzt.

For temporal structures as well as for object declarations the detailed and precise description of performance requirements as a step preceding the representation of knowledge entities constitutes actually a more promising approach than aiming at the declaration of "complete" or "universally defined" concepts.

The third type of temporal reasoning (3) shows the ideal case of merging linguistic - with system performance criteria. Usually, a person asking questions to the system is supposed to associate one institution with one location and building (of course, the inverse relation is not true: it is certainly possible to find several institutions located in one building at the same time). This can be taken as an argument for treating the relationship between an institution and a building as originally or typically time-invariant, although one should provide means for representing the change of residence of an institution as soon as additional temporal mapping information is introduced. From the linguistic point of view, we find even stronger evidence when comparing the following sentences:

* The offices were accommodated in the town hall.
Then they were accommodated in the theatre building.

sounds a bit odd, whereas  

11 (It was) built in 1879.  

12 Having exhausted themselves in a project meeting... the participants took off for an excursion...  

13 From 1883 on, the building accommodated offices, and since... 1968 it has been used as a catholic church.

14 For some difficulties of this kind see e.g. [Holub et al., 1985].

221
The offices were accommodated in the town hall. From 1918 on, they were accommodated in the theatre building.

seems a much better combination. In the case of "accommodating" and other examples like names or the arrangement of furniture in an apartment, it is reasonable not to provide temporal restrictions (e.g. intervals) automatically, but only if explicit indications are to be found. (Manfred Pischal - pers.commu.) One possible realization might be a realization of the relevant attributes as sorts in the hierarchy (e.g. [Bateman et al., 1989]).

2.1.3 Epistemological Adequacy of Definitions

One of the most important criteria for selectional decisions concerning the explicitly represented components of concept definitions is the expected focus of the questions. In our example

Wo befindet sich die Kunstakademie? 15
the answer should concentrate on the information with the highest value for the person asking. Consequently, we are dealing with a predefined search space, restricting completeness of definitions in a philosophical or linguistic sense. A kind of expected focus has to be implemented directly into the knowledge base. It obviously does not make sense to deliver details about the architecture of the building or sophisticated geometric descriptions of the surroundings as a reply. In our given setting we have to keep in mind the goal of the information retrieval process realized in the dialogues. A tourist asking questions to the system will probably try to figure out what kind of places he wants to visit, how he can get there or in which region of the city he can find several alternative sightseeing objects situated closely together. Thus, taking selectional decisions about the number and kind of information that is likely to be asked leads to an implicit coding of the presupposed situation of communication.

2.2 Domain-oriented Decisions

The differentiation between two levels of the domain model for the LILOO system was inspired by [Manz et al., 1985], a discussion of the model of the Jaanus system [Seidemeier et al., 1984]. This distinction is not a formal one but

"For convenience we make an informal distinction between the more abstract concepts and those specific to a particular subject." (Manz et al., 1985)

The distinction reflects an intention posed in (Arens, 1985):

"The designers of the Jaanus system intended that the upper model be general, in the sense that it could be extended to include the concepts of any specific. Such an extension is referred to as a lower model. We expect that lower models will be created by experts in the domain in question." So the upper part of the model can be characterized as rather independent of a specific domain. 16

In the following we will refer to the relatively domain independent part of the model as the Upper Structure, and the relatively domain dependent part as the Lower Structure.

2.2.1 The Upper Structure

The Upper Structure of the ontology portrays some generalized schemes of organization of relative domain-independence. We decided to discriminate between ENTITIES and VIRTUAL CONCEPTS. This distinction aims at isolating the ontological definitions supposed to be the goals of inferring processes from abstract concepts serving at the definition of (a) spatial environments for objects, (b) spatio-temporal environments for events, (c) temporal variability of single features of objects and (d) other specifications (like names, units or names). Thus ENTITIES can be further divided into OBJECT-ENTITIES and EVENTS, with several types of connections between the different resulting parts of the ontology.

Some of the problems arising from a subsumption hierarchy structuring semantic space will be outlined in the following.

[Winston et al., 1987] propose a grouping of five different semantic types of part-whole relations, i.e. component-integral object (e.g. pedal - bike), membro-collection (e.g. ship - fleet), portion mass (e.g. slices - pie), stuff-activity (e.g. steel - cast), feature-activity (e.g. paying - shopping), and place-area (e.g. everglades - florida). In addition to the question whether the conceptual distinction mentioned above proves to be sufficient for the given domain or not, one has to deal with the problem of fitting those relations into a logical formalism with restricted expressive means. In our case, we can make use of three different kinds of ordering mechanisms:

- sets and subsets of sorts, (i.e. the logical subsumption relation)
- two-place predicates delivering additional part-whole relationships, attached to specific sorts and - in the ideal case - selected according to systematic groupings, and
- rules serving for the axiomatic definition of these connections and interrelationships between the sortal expressions, thus determining the density of the represented knowledge.

This problem can be situated in a field of inquiry which is only at the starting point of being considered a fundamental domain of research within AI. As soon as the knowledge to be represented transcends limited domains (e.g. compact technical systems or pre-defined administrative proceedings), you are confronted with an enlargement of modelling decisions without self-evident or correct solutions. In principle it seems inadequate to assume a canonical form of internal conceptualization, as Lakoff pointed out in (Lakoff, 1987):

"Human beings do not function with internally consistent, monolithic conceptual systems. ... Many functioning scientists, in their everyday work, depend on the ability to shift from one conceptualization to another ... There is a distinction between conceptual systems and conceptualizing capacities. The same capacities can give rise to different systems in the following ways:

First, highly structured preconceptual experiences may be different. ... Second, since experience does not determine conceptual systems, but only motivates them, the same experiences may provide equally good motivation for two somewhat different conceptual systems. ... Third, the same basic experiences and the same conceptualizing capacity may still result in a situation where one system lacks a significant concept that another system has."

Following Lakoff, you find equally good alternative conceptual systems which can - moreover - differ in their performance capacities. When looking at the text types in question, we find descriptions of the domain following clearly distinct modalities or linguistic patterns. Consequently, we get a partly redundant coding of e.g. -

15 Where’s the academy of art?
16 A discussion of this distinction within the LILOO system can be found in [van Boeck et al., 1988].
17 They are named roles and features in the formalism used. In our case basically axioms of first order predicate logic.
on the one hand - the time at which a certain building was established, and - on the other hand - the information that the event of establishing a building takes place at a certain date.

In addition to these simpler cases we find as well specifications embedding either object-like or event-like knowledge items, but which do not necessarily appear in both parts of the ontology. Thematic roles, for example, are supposed to capture both the notion of participants in a certain process and of associated circumstances, and the process of building an edifice presupposes the existence of construction plans. This sequence depends exclusively on the conceptualisation of events, whereas an architect or bricklayer in the object-oriented part of the ontology can be characterized by functional specifications without taking into account the sequence they are involved in.

The third aspect mentioned by Lakoff can be compared with the performance of inferences which strongly depends on the kind of clustering of relevant data necessary for solving a task. So for different inference tasks it seems plausible to have special organization principles and clusters of one piece of knowledge by doubling parts of it. On the other hand, preferences of a certain organization of concepts are determined by the properties and capabilities of the inferential processes operating upon these clusters.

2.3.2 The Lower Structure

Following the outlined system of categories in the Upper Structure, we have to classify concepts relevant to the domain. One important difference between Upper Structure and Lower Structure can be found in comparing the depth and degree of changes resulting from modifications undertaken during the modelling process. Fundamental concepts have to be used in many other concept definitions throughout the system, and variations in the Upper Structure will be transferred to other concepts by inheritance properties of the formalism. Specific knowledge has to be coded along the lines given by the categories of the Upper Structure, seems to be more local as to its impact to the overall model, and is intertwined by means of qualitatively different types of connections to other knowledge.

On the level of the Lower Structure the range and variety of decisions appears at first glance much smaller. Some questions are still left open, though, even if criteria of domain dependence, task orientation and compatibility with language surface structures are seriously taken into account. A typical example is the conceptualization of events. There are two major possibilities of creating complex structures out of parts, i.e. for the definition of internal coherence of subevents. First, you can distinguish patterns of events determined by a cultural environment and consider their decomposition into single activities almost independent of any cultural setting. So MEETING someone basically consists of MOVing towards him, STOPping at him, SAYing something etc. In this case, we can establish an analogy with concepts of materialization of objects like CAR, STEEL, given an important qualitative difference between the two levels. In the second variant, you can postulate that there is no qualitative difference, but instead the option of zooming up or down to a level of optimal grain size. Decisions concerning grain size have to be taken according to the application of a system and the level of knowledge that the representation is supposed to capture.

Decisions of this kind modify to a greater or lesser extent the kind of inferences that finally can be drawn (by means of inheritance mechanisms or explicit rules) as well as the resemblance to a lexical realization of the concepts. It should be noted here that two important parts of the ontology concerning objects and events do not exclusively correspond to nouns and verbs, respectively: a WORKSHOP or a WALK should certainly be represented as events. So, as you would expect, the domain model does not reflect in a direct way language surface phenomena. But, compatibility with language surface structures has to be ensured by taking into account the linguistic processes translating natural language utterances into a logical form. Especially lexical entries are in our approach crucial for the correlation of words and conceptual meaning.

Modelling decisions in the Lower Structure have to be validated in a sense of exploring their appropriateness for a given, restricted application. Considerations of system performance in this context lead to two types of requirements. We already mentioned compatibility with language surface structures. On the other side conceptual appropriateness for the expected questions has to be ensured. As a central specification of this second point, the set of relevant entities for a specific Lower Structure must be determined. The criteria for relevance within this context are on the one hand in some sense specified by the task of the system (a.o.) and the content of the texts. Obviously, this task is even less sharply outlined than e.g. in natural language interfaces to databases, where user expectations of the overall system performance are guided by the content and purpose of the database. Therefore only a vague anticipation of user interests (in our context utility for the plans and goals a tourist is expected to have) may serve as a guideline.

On the other hand, one has to face the question what happens in those cases where the user's questions either transcend the coded knowledge items or move towards aspects apart from the conceptual core of the model. Consequently, one of the measures of validity for the Lower Structure consists of a lower degree of semantic density in the knowledge base for peripheral topics. So, e.g. the representation of alloy consists mainly of the link to road. For a question concerning alloy the model would deliver this aspect, but no information about the typical distribution of trees on both sides or about the well-suitedness of such a road as a place for driving or walking around.

3 Further Aspects

The domain model sketched here plays a significant role within the process of text understanding and answer generation in the LILOG system. Therefore, several linguistic aspects have to be taken into account. A first approach in that direction is quoted in [Besteman et al., 1986]:

(We assume) "that differing concepts in the upper model will differ in linguistic realization... Subordinating a concept from an application domain to some particular concept in the upper model strongly constrains the range of linguistic realizations that are available for that concept."

In our approach we want to avoid consequences of class assignment in that sense. Instead, we have to attempt adequate groupings according to domain, task, and linguistic performance criteria.29

In some cases, we even have to deal with conflicting requirements from domain-dependence on the one hand and linguistic orientation on the other. On the conceptual level, there should be one single representation of the charged one has to pay when visiting an institution like a museum, containing specifications like reductions for students, etc. On the surface level, you may find a couple of ways to deliver this information, e.g. The charge is ..., You have to pay ..., and The entrance fee ...

A special case of the difference between domain oriented models and linguistic phenomena arises within the problem of handling adjectives. According to M. Fishel (pers. comm.), adjectives are a coherent syntactic class, and following semantic classification schemas in conjunction with domain specifications, one has to consider different subclasses with specific representation demands. So at least two different groups of adjectives can be distinguished, according to their 'reference' use or calibrating semantic content. Examples of the first group are modern and international. This type

---

29While resembling roughly the distinction of epistemology and grammar in [McCarthy and Hayes, 1969].
of adjectives can be represented by one-place predicates or – in our case – by concepts. Exclusive can be analyzed as an adjective for ranking without having an explicit scale connected to it, while big alludes at a context-dependent measurability. Both belong to the group of calibrating adjectives and should be represented by a rule system aiming at an axiomatization of the context-dependent truth value.

It should be mentioned, though, that actually no empirical or theoretical solution concerning the relation between linguistic competence and knowledge of the world is available. As a consequence, quite a number of decisions that have to be made during the process of designing an architecture of a natural language understanding system remain exclusively motivated by system performance criteria. One of these possible shifts consists in transferring the resolution or explication of meaning from the knowledge base and inferring component to linguistic components like explicit lexical coding or processes on the basis of a surface-oriented semantic reconstruction of a natural language utterance.21

4 Conclusion

The task tackled in this project belongs to the area of common sense modelling. For text understanding we need "... millions of facts and heuristics that define ... a machine-understood embodiment of twenty century commonsense reality."[Lenat and Guha, 1988]

In our case, restricting ourselves to a limited domain, we don't need those millions. But the task of ontological engineering as "... defining the categories and relationships of the domain"[Lenat and Guha, 1988]

means focusing the necessary extension of the scope of (formal)
theoretical conceptualizations.

"This is empirical, experimental engineering, as contrasted with 'ontological theorizing' that philosophers have done for millennia."[Lenat and Guha, 1988]

In this sense we hope to contribute some promising solutions, making explicit some of the tacit prerequisites for successful text processing. Among the criteria for success, arguments of cognitive adequacy can be said to play a crucial role – at least if one follows the premises of [Link, 1988] that even in human communication you find optimal degrees of depth or superficiality.

References


21 Recent works in e.g. Cognitive Linguistics show that this field is receiving growing attention.


DISTRIBUTED OBJECT-ORIENTED INFERENCING IN ADA

Richard Simonian
Harris Space Systems Corporation
295 Barnes Blvd.
Rockledge, FL 32955
407-633-3800

ABSTRACT

This paper describes the Ada Inferencing Mechanism (AIM) which was implemented using an object-oriented preprocessor for Ada. AIM effectively employs Ada’s powerful tasking mechanism to decentralize inference control. The object-oriented paradigm used to design and develop AIM provided a means to create an expressive, efficient, and flexible expert system toolkit which can be embedded as a component of a larger system or exist as a stand-alone utility.

1. INTRODUCTION

Ada presents several unique advantages as an implementation language for artificial intelligence. Ada supports and enforces sound software engineering principles; this will require AI components to follow the same software lifecycle expected of traditional components. Ada has a built in concurrency construct with the tasking mechanism which allows parallel algorithms to be built entirely within a standard language. Ada is designed for embedded, real-time systems, a capability which symbolic languages cannot match. Finally, Ada is a government-mandated standard with the goals of enhancing interoperability, productivity, and maintenance of extremely large software systems. If Ada can be effectively used to develop artificial intelligence, then AI components can become well-integrated, useful components of complex systems.

Previously, the major obstacles in writing such a system in Ada have been Ada’s strongly-typed, static nature and the fact that Ada lacks the facilities of a true object-oriented language. To overcome these difficulties, the Ada Inferencing Mechanism (AIM) described in this paper was implemented through the use of an object-oriented preprocessor tool developed at Harris called InnovAda [7]. InnovAda allows a user to interactively define classes in a multiple-inheritance hierarchy along with their associated methods. The output of InnovAda is compilable Ada source code. Two other similar preprocessor tools for Ada exist [9] [2].

2. THE ADA INFERENCING MECHANISM

This paper presents the design and implementation of the Ada Inferencing Mechanism [8] (AIM), a parallel, object-oriented expert system development toolkit written entirely in Ada. By extending the power of Ada with object-oriented facilities provided by InnovAda, a unique expert system architecture has been developed. AIM has several key features:

1. It is implemented entirely in Ada.
2. Rules are asynchronous objects which can request initial data, forward chain, backward chain, or operate opportunistically.
3. AIM parses, interprets, and executes rules at run time.
4. The user may specify a structure of contexts, each of which may have associated rules and parameters.
5. Contexts operate asynchronously and communicate through layers of shared blackboards.

3. ARCHITECTURE

![The AIM System Architecture](image)

Figure 1: The AIM System Architecture

The top-level system architecture for AIM is shown in Figure 1. The knowledge engineer constructs a system composed of a
hierarchy of contexts, each of which refines its parent’s domain in some fashion. An AIM system can be viewed as a community of cooperating experts. Each context has a specified domain, maintains its own working memory and goal blackboard, and has some number of rules and sub-contexts.

4. DISTRIBUTED CONTROL

AIM has no centralized inference control; rather, control is distributed across all the rules. Each AIM rule is specified by the user to be of type Forward, Backward, Opportunistic, or Initial. Therefore, each rule, by virtue of its class, knows how to control its behavior. For example, backward rules may respond to goals posted within their area, while forward rules are driven by relevant data postings. Rules are parsed and interpreted at run-time. New rules can be defined and inserted into the system at any point. Each rule is implemented as an Ada task and is grouped with a specified context.

Contexts are also asynchronous. Their main function is to control communication throughout the system. Contexts are built in a user-specified hierarchy and communicate through levels of shared blackboards. Each context maintains its own working memory. As a context receives goals and facts, it may decide to propagate the information to higher and lower level contexts, each of which may decide to reject or accept the information, based on the information’s confidence. Both the working memory and the goal blackboard are accessed by rules of the appropriate type. Backward rules monitor the goal blackboard, forward rules monitor the working memory, and opportunistic rules monitor both.

Contexts operate independently of each other as do the rules; therefore, no determinism or synchronization should be assumed. Parameters in the rule conclusions may have associated Mycin-style confidence factors to specify how certain the designer is that the conclusion can be made given the evidence in the antecedent.

AIM parallelism was also designed to be taken one level higher. Just as AIM’s contexts cooperate asynchronously to solve problems, multiple instantiations of AIM could be executed on multiple processors. The message passing capability of InnovaAda facilitates the removal of proximity issues. A message passing operation could be modified so that it occurs across a network; the node would be transparent to the sending and receiving objects. Thus, a complex array of AIM processes could be built, capable of reasoning about very diverse domains.

5. AN OBJECT-ORIENTED APPROACH

As stated earlier, AIM was implemented through the use of a preprocessor tool called InnovaAda. AIM is composed of approximately 30 InnovaAda classes, their methods, and several supporting Ada packages.

As a result of AIM’s object-oriented design, the inferencing capability can easily be integrated as a component of a larger system. In addition, the user can build the knowledge base in an object-oriented fashion. The AIM design process follows these steps:

1. Specify the context hierarchy and their domains.
2. Specify the parameters, including the name, type, any default values, and prompt strings.
3. Specify the rule-base in relation to the contexts and parameters.

Each step in the specification process consists of creating a text file with the appropriate syntax. These files are loaded into AIM at run-time. The following sections discuss the details of the major objects in AIM and how they interact.

5.1 Parameters

AIM’s processing centers on parameters. Figure 2 shows the Parameter class semantic diagram. Users post parameters as goals, contexts maintain them on blackboards and in working memory, and rules access, test, and set them. Internally, a parameter maintains a value, a confidence factor, a history for truth-maintenance, and the time of last update. The parameter file is defined by filling out copies of the following template in a file:

```
%desc parm [parameter-name] [parameter-type] default [default-value] prompt ["prompt string"]
```

The parameter type is one of BOOLEAN, INTEGER, STRING, or FLOAT. The parameter’s default may be left blank but, if specified, must be of the type given. The parameter prompt string may also be left blank. If specified, it is used to prompt the user for new parameter values as required by AIM.

![Parameter Class Diagram](image)

Figure 2: The Parameter Class

5.2 Blackboards

Many blackboard systems have been developed since the original Hearsay project, including at least one in Ada [11]. The key advantage of blackboards is that they provide a fairly elegant mechanism of cooperation between asynchronous knowledge sources. However, a disadvantage stated in [10] is that “the agenda that controls the flow of triggered knowledge sources may become a computational bottleneck.” Studies have been performed [3] which looked at the possibility of adding control reasoning to a blackboard system to mitigate the problem; however, this really just pushes the problem a bit further over the horizon. AIM uses blackboards, but eliminates centralized control entirely. The semantic diagram for the Blackboard class is shown in Figure 3.
AIM blackboards are used to communicate goals between levels in the context hierarchy. Each context has a pointer to its blackboard, which is a subclass of CriticalRegion. Therefore, blackboards safely allow updates and accesses from multiple sources, providing that the source correctly enters and leaves the critical region.

AIM blackboards serve a function similar to that of traditional blackboards. Traditional blackboards, however, are tightly integrated with their problem-solving environment in that each knows where it exists in a hierarchical structure, and who has access to it. AIM blackboards are more isolated: they add data items to their internal structure, and respond to inquiries regarding their contents. They do not care who has access to their data. The working memory of typical blackboards is somewhat separated from the AIM blackboard; it is represented by a Parameter binary tree.

Rules are asynchronous knowledge sources which monitor a blackboard. Rules of type Backward or Hybrid monitor their context’s blackboard for relevant goals. Rules may also direct their context to post new goals at various levels in the hierarchy. AIM blackboards provide no focus-of-control or agenda; rather, they enhance the parallelism of the system by serving as an asynchronous communication point between contexts.

5.3 Rules

Rule definition requires one to follow the grammar for antecedents and consequents. The semantic diagram for the Rule class is given in Figure 7 and the rule is created by filling out copies of the template shown in Figure 4. In the antecedent, as shown by the grammar in Figure 5, the user has the option of testing parameters through many combinations of conjunctions, disjunctions, and nested expressions. The consequent, as shown by the grammar in Figure 6, allows the user to set parameter values, perhaps with confidence factors, output strings, read input from the user, and retract information.

```
defrule RULE-NAME RULE-TYPE
   <= comments allowed anywhere
      (antecedent)
   =>
      (consequent)
%end
```

Figure 4: Rule Definition Template

antecedent: (, condition ,).
condition: {NOT} comparison;
{NOT} condition, junction, {NOT} condition.
comparison: exp; exp, relation, exp, (, condition, );
          KNOWN exp.
exp: term; term, weak op, term.
term: element; element, strong op, element.
element: const; variable; (, exp, ).
const: digit; const, digit.
variable: letter; variable, letter; variable, digit.
relation: =; <=; >=; <; >; =/=.
junction: AND; OR.
weak op: ++; --;
strong op: *; /.

Figure 5: Antecedent Grammar

consequent: (, postcondition, ).
postcondition: (, action, );
             (, action, ), junction, postcondition.
action: ’RETACT’ parameter;
       ’OUTPUT’ string;
       parameter := exp1;
       parameter := exp1, ’CONFIDENCE’,
       numeric_literal.
exp1: ’READ’;
exp.

Figure 6: Consequent Grammar

5.3.1 Rule Types

As previously stated, there is no centralized inference strategy in AIM. The object-oriented approach allows rules to define how knowledge is processed in the system. When the user creates the rulebase, he or she associates with each rule a RULE-TYPE, which is one of:

1. BACKWARD. The rule will operate in backward mode. When a goal is posted (to the local goal blackboard) which the rule contains as a parameter in its consequent, the rule will attempt to resolve that goal.

2. FORWARD. The rule will operate in forward mode. When a working-memory update occurs to a parameter which is contained in the rule’s antecedent, the rule will test itself and possibly fire.

3. BOTH. The rule will operate opportunistically; that is, as both a forward and backward rule. Working-memory updates as well as goal postings may affect the rule.

4. INITIAL. The rule will be tested and possibly fired only when the rule is initially loaded. These sorts of rules are generally used to query the user for initial and required data.
5.3.2 Rule State Switching

In order to make rule objects asynchronous, Ada tasks are used. After a successful rule parse, a new task is initialized. Each instance of a rule task has a rule object instance, a parameter tree pointer, a parent context, a certain mode type (one of Backward, Forward, Both, or Initial), a pointer to a goal blackboard to monitor (for Backward or Both), and a specified cycle time (which defaults to one second). A rule task operates in a continuous loop until a Stop fact is posted.

Forward rules are quite straightforward; backward rules have a bit of a twist. A backward rule operates as a finite state machine with two states: Initial and waiting. Initially, the rule just scans the goal blackboard for goals which it might be able to solve. If any exist, the rule determines any unknown parameters it needs. These are posted as subgoals to the context and added to the rule's list of waiting-parameters. The rule is then switched to waiting-state. At this point, it has effectively become a forward rule, since it will fire when its list of parameters are solved. Note that when the rule is waiting, it can ignore any other goals which might come in since they will be implicitly solved. When the list of waiting parameters is empty, the rule is tested and possibly fired, and switched back to initial-state.

This scheme of switching states for backward rules has several advantages. First, it removes the usual boundary between forward and backward rules. Traditional expert system have some default reasoning mechanism, such as backward reasoning in Mycin or forward reasoning in OPS5. There are methods to simulate one by using the other, but this is not efficient or elegant. AIM has no default reasoning control; the user simply specifies at the rule level how processing should occur.

Second, the technique implicitly stores rule activation hierarchies which usually must be explicitly maintained. An example is shown in Figure 8. The activation hierarchy shown in this figure is implicitly stored by AIM via this rule-flipping scheme. In this example, when \( u \) and \( v \) become known, Rule 4 fires and reverts to backward mode, and the chain slowly unravels.

Third, the state switching appears to implement a sort of dynamic Rete network [4] for backward rules. Further research on this topic would be useful.

In Figure 8:

- R1: A, B <= X, Y
- R2: C <= Y
- R3: M, N <= Y
- R4: Y <= U, V

QUERY A:
1) R1 switch to Forward
2) Query X, Y
3) R4 switch to Forward
4) Query U, V

5.4 Contexts

The Context class is the most complex class in AIM. If you were to view the integration hierarchy of AIM, where higher-level objects are composed of simpler objects, Context would be at the top. The Context class semantic definition is shown in Figure 9. Context instances respond to messages to post or remove facts and goals, load a rule base or parameter definition file, maintain their parent and child contexts, and answer queries for specific parameters.

The user specifies the AIM context hierarchy by creating a text file with any number of the following templates completed:

```
%start context CONTEXT--NAME
    parent PARENT--NAME
    domain DOMAIN--PATTERN
    clock--cycle CYCLE
%
```

The parent, if left blank, defaults to none; in other words, the root context. Only one root context can be defined for an AIM hierarchy per session. The domain pattern is a regular expression (as specified in [11]) indicating the domains which this context applies to. If left blank, it will default to "??", or all domains. The domain pattern is meant to be used to direct the transmission of parameters and goals throughout the system. The cycle time is a floating point number indicating how often the context and its asynchronous rules will update. If left blank, the cycle time defaults to one second.

Within the processing for the INIT procedure in rule tasks, an asynchronous Context pointer is used. Asynchronous objects are created by Invoking Ada by default for each class; they are simply task types whose entry calls correspond to the methods defined for that class. Using an asynchronous object for Context instances is ideal because a context must be shared between many parallel rules. The Ada tasking mechanism automatically queues simultaneous entries to a task; therefore, if two rules are trying to post goals to the same context, their entries will be handled correctly. This would not be the case, without some sort of locking mechanism, with sequential objects.
5.5 Object Interaction

Now that the major components of AIM have been defined, let us reiterate how the different rule types interact with contexts. Figure 10 diagrams the component and access relationships between contexts, blackboards, parameters, and rules. Each instance of a context has its own working memory, represented as a binary tree of parameter instances. It also has an instance of a goal blackboard object. A context has no references to its rules; however, rules loaded into a context have pointers to their context. They also have direct pointers to the goal blackboard and the parameter tree.

```
Context
  Goal Blackboard
    Forward Rule
    Hybrid Rule
    Backward Rule
  Working Memory
```

= component

\[\text{Figure 10: AIM Object Interaction}\]

6. FUTURE WORK AND CONCLUSIONS

The groundwork for a parallel, object-oriented inference engine in Ada has been laid. The front-end developed for the tool exercised the architecture and proved the concept; however, it by no means achieved the potential of AIM. There are several areas where further work could be done to improve the tool. Possibilities include more effective use of the context and parameter domains, addition of daemons, and the addition of a unification algorithm.

The work on AIM has lead to several new concepts, most of which show solutions to the problem of implementing artificial intelligence in Ada. The major contributions of this research are:

1. The development of a unique, concurrent, object-oriented model of inferencing in which control is distributed to the level of asynchronuous rules.

2. The development of a method for treating forward, backward, initial, and opportunistic rules simply as different states of a generic rule.

3. The demonstration that a preprocessor tool such as InnovAda can be used to provide Ada with true object-oriented capabilities. Ada's numerous advantages (e.g., its enforcement of good software engineering techniques and its built-in concurrent processing mechanism) can now be brought to bear on artificial intelligence.

The actual execution of the Ada Inferencing Mechanism (AIM) developed for this project showed that the use of Ada's tasks to distribute inferencing is feasible, even on a single-processor machine.

7. REFERENCES


Expert Explorer (XX): An Assistance System for Hydrocarbon Exploration

James C. Bezdek
Div. of Computer Science
University of West Florida
Pensacola, FLA 32514

Gautam Biswas
Computer Science Department
Vanderbilt University
Nashville, Tennessee 37235

Robert Cannon
Computer Science Department
University of South Carolina
Columbia, South Carolina 29208

Christopher Kendall
Geology Department
University of South Carolina
Columbia, South Carolina 29208

ABSTRACT

The Expert Explorer (XX) software system is an integrated set of computer programs that assists exploration geologists in the evaluation of potential oil producing regions ("plays" and "prospects"). The system is currently comprised of database, pattern recognition, basin simulation, expert system, and report generation components. This paper summarizes the current state and capabilities of the system; and discusses ongoing research and development issues that impact hydrocarbon exploration, especially as it relates to current efforts in the Gulf of Mexico (for example, the Sunnyland and Smackover fields). In this report we emphasize the role of the expert system component for database characterization and construction of the analysis kernel (the attribute matrix).

INTRODUCTION

Expert systems as potential diagnostic and decision support systems in industry in general, and in hydrocarbon exploration in particular, have received a great deal of attention in recent years (Miller 1986). For example, some of the better publicized expert systems for geological use are PROSPECTOR, designed and implemented by SRI International to evaluate mineral exploration; INIX and DIPMETER ADVISOR, designed by Schlumberger Doli Research; MUDMAN, designed by NL Baroid and NL Industries; and mu PETROL, a system for basin classification developed at the U.S.G.S. DIPMETER ADVISOR and MUDMAN are both being used successfully in the petroleum industry today. Mu PETROL and INIX are still under development and are being tested (Kahn and McDermott 1988).
Despite extensive financial support and development, expert systems within the oil industry have had a checkered history. This is due in part to overpromise by developers, which in turn led to unrealistic expectations but few significant returns. One major problem with current systems is that they are based on very narrow geological models, and the experiential knowledge of a human expert. This results in wrong inferences when systemic reasoning is applied to situations which are at variance with the context in which development took place. For example, the human expert may be very proficient at analyzing North Sea prospects, but much of this expertise may not apply to prospects in the Gulf of Mexico.

A second problem in building expert systems for oil exploration is that there has been a decided lack of effective communication between the designers and users of such systems. Modeling geological expertise is especially difficult because of the highly intuitive and ill-defined nature of exploration. Thus, knowledge acquisition in this domain, the key to building a successful expert system, requires a good understanding of the problem, concepts and terminology of the domain, and of the problem solving methods of the expert(s) involved in system design. System designers have often been reluctant, unwilling or even incapable of devoting sufficient time and effort to these tasks, the end result being systems which have less than expected levels of robustness and utility.

A third problem is the data, which may be: incomplete, inconsistent, inaccssible, and non-uniform in both quantity and quality. Data for a potential oil-bearing area ("play") are obtained from various quantitative sources such as seismic cross-sections and well logs; and qualitative sources such as outcrop data, field notes, and maps. Large computer programs for analyzing numeric data already exist (the oil industry is today the largest industrial user of supercomputers). These programs provide the best quantitative data about a play. Those results, together with all of the other data collected about a play fill corporate databases. Unfortunately, the amount of data is so great and diverse, and so poorly encoded and documented, that it is usually inaccessible to explorationists for the purpose of play analysis; thus, exploration decisions continue to rest on the abilities of a few human experts.

A fourth major problem with many current designs is that uncertainty is often ignored, despite the fact that it is both inherent and pervasive in oil exploration. For instance, there may be errors in numerical data gathering, either due to faulty instruments or the recording mechanism. Explorationists resort to many rules of thumb when reasoning by analogy; each of these is accompanied by uncertainty. Similarly, rules from which conclusions are drawn may be based on incomplete and/or inconsistent data, so that the rules themselves should have certainty factors associated with them. It is very important, therefore, to incorporate approximate reasoning in the expert system framework. XX incorporates this form of inference. Our model reflects geologic uncertainties through a scheme of weighted production rules, semantic representation of relational knowledge between basin model variables, and linguistic approximations to represent properties of various objects involved in the analysis.

Before making a recommendation to drill for oil, the explorationist must "interpret" the data, i.e., make an analysis of its potential to produce oil. At this point the experienced oil explorationist is at an advantage because he or she has interpreted other plays. The expert relies heavily upon his or her knowledge of other plays, drawing analogies whenever possible. All kinds of data may be used—perhaps a map of an area, a prototypical well log from another area, or knowledge of the presence or absence of certain flora and fauna. These may be all in the analyst's memory, on a shelf or in a file cabinet in the office, in the library down the
hail, or buried in a corporate database with little or no retrieval facilities to any but the computer expert. Analogs help the analyst compare the new play to those with which it shares common features. By knowing the exploration and production history of analogous plays, the analyst can make a better judgment regarding the potential success of the new play.

Rather than design an expert classification tool, it seems more useful for oil companies to have a knowledge-based system that facilitates data encapsulation, and which provides "navigational assistance" in acquiring salient data and information from all sources (including experiential knowledge), reports and maps. Our system helps users follow a line of analogical reasoning in characterizing a target play in a manner similar to that of a human expert, and allows the user to access known exploration data in the casebook to compare them with target play data. Moreover, the system records and imbeds the associational knowledge of experts that links features observed in the data (e.g., seismic, well logs, etc.) to relevant play or prospect characteristics. Thus, the system provides a way to capture and represent the geological parameters considered significant for determining oil bearing potential. Another useful facility incorporated by our system is its ability to simulate sophisticated geological models which trace the depositional history from which inferences can be made. Finally, our system provides an integrated approach to database management and interactive graphics displays and some inferential capabilities with respect to potential hydrocarbon plays.

A final use of XX pertains to the retention of corporate knowledge and strategy. With the current downturn in the oil industry, little domestic exploration is being conducted. Retiring explorationists frequently vacate exploration and research offices, leaving the burden of play interpretation to inexperienced geologists. Thus, there is a very real danger in some companies for a "knowledge vacuum" to form. XX enables users to capture the expertise of company explorationists by allowing them to customize the rule base that builds the system kernel. Corporate strategy is captured when the system designer places different weights upon the attributes being considered, thus influencing the determination of analogs so as to emphasize those which a particular company considers most important.

**SYSTEM COMPONENTS**

Figure 1 shows the architecture of the XX system. The major modules are the **Graphics Interface, Database Manager, Field Database, PLAYMAKER, SEDPAK, CLUSPAK, and REPORT**. SEDPAK is shown linked to the system with a dotted line because it currently runs as a stand-alone auxiliary to the XX system. Users interact with the system through the System Manager (not shown), which allows selection of various system functions by use of pop-up windows and a mouse. The **MONITOR** in Figure 1 represents the host environment. XX currently runs under UNIX on any engineering type workstation, that includes the SUN 3 and 4, DEC 3100, and APOLLO DN 3000. Graphics functions are implemented in X-windows (Scheifler and Gettys 1986) and the code is primarily written in the C and Common LISP programming languages. The system is not a monolith. Rather, each of its components (system manager, database manager, PLAYMAKER, CLUSPAK, SEDPAK, and REPORT) executes as a separate Unix process, thereby minimizing memory demands made by foreground processes (Bach 1986). Thus, the components of the system can be developed and tested autonomously, making system integration much more straightforward than if the system were developed as one large program. The system manager (not shown) is currently being rewritten using the MOTIF environment, which facilitates fast interactive graphics, multi-windowing, menu handling and inter-modal data transactions.

The startup screen for the XX system utilizes pop-up menus. The user can choose to initiate a description of a new play, or to continue to work on an existing play by selecting the appropriate function with a mouse. Each selection causes a new menu to open so as to allow selection from the subset of functions associated with the major function. In this mode no keyboard activity is needed unless actual descriptive text is to be entered or a report is to be written. Processes for the secondary functions of the XX system are already executing, e.g., the Database Manager, or may be forked, e.g.,
CLUSPAK. The interface between processes is accomplished by sockets (Bach 1988), so that messages can pass between processes asynchronously. Host systems are designated for resident XX application and database files. The X-Window System provides remote access to the host system from any location. Our goal has been to make the system as modular as possible, enabling further enhancements to one component with as few side-effects as possible.

**Attribute Matrix (System Kernel)**

The primary repository of expert knowledge is in the form of rules as they apply to the attribute matrix (the system kernel) A in Figure 1. Each column of A corresponds to one dry well or producing oil field, or to a target (undeveloped) play or prospect, whereas the rows of A are associated with basin properties that are exploration keys. The play is characterized by data about the reservoir, its source, and its seal. The reservoir is the actual rock in which the oil is located; the source is the rock from which the oil is thought to originate; and the seal is the rock which causes the oil to remain trapped in the reservoir. Figure 2 depicts the hierarchical structure of one portion of A.

The attribute matrix resides in a relational database, and is linked to a rule-based system called PLAYMAKER that helps expert geologists build each play column (hence "PLAYMAKER"). Currently there are 12 major keys. Each major key is further decomposed into a number of minor keys (or attributes) as shown in Figure 2, and each minor key in turn may possess a number of values. Each value defines a row in the attribute matrix, and in each column of such a row one finds the value of a numerical, linguistic, or logical variable. In Figure 2, all of the attributes of the property (sediment type) are identified semantically, and each is assigned a numerical value, as would be the case for each column of A. Except for image and textual data (items such as a general description of and notes about the play), all attribute values are represented by a pair \( (v, e(v)) \), where \( v = \) value, and \( e(v) \) is a number in the closed real interval \( [0,1] \) which represents the degree of belief in the particular value being associated with the attribute (0 for not present; 1 for fully present). Thus, in the example of Figure 2, the reservoir for the field being entered is believed to contain no gypsum, salt or pyroclastic material; and is formed predominantly by conglomerates. Note that the sum of values beliefs for a given numerically defined property is constrained to being less than or equal to 1, this being a requirement of the Dempster-Shafer theory that is used as the framework for evidence combination and computation of beliefs. Demographic and gross basin characteristics are entered by a user for each desired field or play. Some of the values have been given fixed weights (influence coefficients) by the expert geologist (system mentor); while others are derived by PLAYMAKER using the imbedded, weighted production rules and considerations related to the quality of input data available to the system users. For example, the presence of salt in the reservoir may preclude the occurrence of coal; PLAYMAKER knows this, and uses this to compute beliefs for the related individual extent(s) automatically.

Depending on the needs and desires of the user, the attribute matrix might contain all the plays in a geographic area, such as the North Slope of Alaska; or it might include all of the major oil fields of the world. Currently, there are about 200 attributes which possess a total of more than 350 values, so conceptually (A) is about 350 \( \times 40 \) in size at present. The baseline system supplied to XX industrial sponsors currently contains the following (Gulf of Mexico) fields in columns of the attribute matrix (A): **Jurassic** (Jay, Smackover, Pachuta Creek, Walker Creek, Red Rock, Pool Creek): **Lower Cretaceous** (Calhoun, Black Lake, Kerlin, Sunnyland, Delhi, JFS, Waveland): **Upper Cretaceous** (Kurten, South Calton, Big Wells, Sabine Chalk): **Paleocene Eocene** (Hazel Creek, Katy, Rusita, Vallentine and S. Halletville, Wilbur, NW Thompsonville): **Oligocene** (McAllen Ranch, Old Ocean, Gilis English Bayou, Big Hill, Seeigsen, Tom O'Conner, Port Acres and Port Arthur, South, Bosco-Duson Ridge, SW Lake Arthur, Turtle Bay): **Miocene** (Vermilion Blue): **Pliocene** (Hitchcock): **Pleistocene** (Chandeleur Sound Block 25, West Whitleake, Bay Marchland-Timbaleia-Cailou Island, S. Timbalier Block 54, W. Boston Bay, Eugene Island Block 330, East, Cameron Block 250, High Island Block A560, High Island Block A560). We anticipate that corporate systems
will eventually utilize several thousand fields (columns) for an online system.

**Database Manager**

The textual and numeric data in the attribute matrix are stored in a relational database, so (A) is actually a flat file, as shown in Figure 1. The Database Manager performs insertions, deletions, and retrievals from the kernel (A). Access to (A) is by SQL queries issued by the Database Manager upon request by other system components (Date 1986). In this manner all database functions are centralized in one module of the system. We emphasize that although our system uses INGRES as the developmental DBM, the system can be interfaced to any DB (e.g., ORACLE) that accepts SQL queries. Thus, the system is transportable to a wide variety of corporate DB structures. Linkage between PLAYMAKER and the DB is complete; linkage to REPORT is underway; and linkage to SEDPAK and CLUSPAK will begin in the summer of 1990.

**Field Data**

The primary source of and repository for field data such as seismic line interpretations, images, well logs, notes, maps, basin history and the like is in corporate databases. Public access data is limited, and what can be found usually comes from scattered and incomplete articles in the exploration literature. Since different companies have diverse storage mediums, DB systems, retrieval capabilities, and corporate strategies, our approach to the problem of interfacing with corporate data has been to build XX modules that issue SQL calls. Diversity further necessitates that each installation of XX be more or less customized to the resident data. Consequently, PLAYMAKER has been designed to build most of the kernel (A) automatically once installation is complete. Since the rule editor in PLAYMAKER enables each user to customize the inferencing structure of PLAYMAKER, each user will eventually have a completely proprietary system. Current plans call for installation of a fully operational beta test version of XX at a host oil company in the summer of 1990.

**PLAYMAKER**

PLAYMAKER provides an intelligent interface to a user who wishes to update or add field data in the attribute matrix. Some attributes in (A) may be synthesized from several properties, data, or pieces of information. The relative importance of each attribute contributing to a derived attribute value is weighted by the expert; and further, the overall belief in each rule is recorded. Different companies will, of course, opt for different rules and rule weights. Thus, the rule editor enables each company to retain its proprietary strategies by allowing them to add, delete and modify rules in the knowledge base as they see fit. The rules and rule editor in XX are briefly explained with an example.

Figure 3 illustrates the rule editor. In the figure, rule 1208 has been recalled (or is under initial construction). Attribute-value pairs for 1208 are taken from the attribute library (or added to it if not present) by selection and clicking. The LHS of the rule represents a conjunction of attribute-value pairs, whereas the RHS represents possible conclusions about carbonate-evaporite facies that can be inferred from the given data.

Rule 1208, supplied to the system by an expert, is read as follows: if the well data possessed by the user has certain characteristics (e.g., wells penetrate the area of interest, etc.), and the depositional setting of the field reservoir was on the shelf, and the well data indicates that the fauna in the reservoir are diverse, then it is very likely that the types of carbonate facies present are buildups, barriers and shoal-ups (shoaling upward sequences). Rules in the system are partitioned into subsets by their context. The context of the partition for this rule is that the system has already determined by interaction with the user that the primary lithologies present are carbonates (as opposed to clastics). The other factors on the LHS of the rule indicate that the user has access to well data that penetrates the area of interest; and further, it is of sufficient quality to enable the user to characterize the fauna. Also, if it has already been established that the area of interest lies on the shelf, then possible carbonate facies are the ones indicated in the figure. Note that this rule is based on the expert's judgment or experience. That is, the
user is allowed to qualify each conclusion of the rule by specifying a belief value for the strength of each conclusion. In this example, the user has expressed a belief of 8/10 in the importance of the LHS evidence (10/10 would imply that the evidence is definitely conclusive). And further, given that multiple conclusions are possible, relative rankings for the strength of belief in each conclusion are recorded. In the example, the two conclusions are given ranks of 3 and 2; this indicates that belief in the first conclusion is "one and one-half" times as strong as belief in the second one.

Multiple conclusions on a line imply that the evidence on the LHS cannot really differentiate between individual conclusions (e.g., between barrier and buildup). The rankings and weights provided by the expert are converted into a belief function in the Dempster-Shafer framework associated with the rule. Details and justification for this methodology are found in (Biswaas and Anand 1989).

Some of the attributes used in the LHS of a rule may have themselves been derived during the process of play characterization (e.g., Use_Well and Dep_Set). Therefore, they may have prior belief values assigned to them from another point in the inference process. In this case, belief in the conclusion(s) of the rule being edited on screen is automatically modified by suitably weighting the rule in question using multiplicative rules for the combination of evidence as described in (Biswaas and Anand 1989).

Contextually, attribute weights (e(v)) are also interpreted as indicating, in some instances, the "quality" of the source data from which they were derived (as opposed to the "extent" to which the attribute is present, or is possessed). Each source may provide varying quality of information on a specific property. For example, seismic data provide better information on sediment geometry than well cores, while well cores provide more reliable information on sediment lithology than does seismic data. Once again, this information is provided by the expert and is readily incorporated into the system.

Data from various sources are often approximations or extrapolations. Sensing devices are placed at various points in an area, so that readings between data points must be approximated. Likewise, well logs are fairly accurate at the data source but must be interpolated between them. Thus, it is fairly common for data to conflict because of the inaccuracies of observation or of approximation. Again, PLAYMAKER resolves conflicts of data by rules which weight data from better sources more heavily than data from poor quality sources.

For example, if the user stated that there was well data in the region of the prospect, but not in the immediate vicinity of the area of interest, PLAYMAKER would use an expert supplied rule that read: If Use_Well is medium and depositional setting is shelf and fauna is diverse, then replace the assigned belief value (8 in Figure 3 = 0.8) by (5 = 0.5). Notice that the belief in the evidence recorded by the expert is lower in the new instance. Therefore, the belief in the two sets of conclusions will be correspondingly lower. Specifically, the belief value (e(v)) associated with shoal-up would be (2/5)•0.8 = 0.32 in the first case (the one shown in Figure 3), but only (2/5)•0.5 = 0.20 in the second.

PLAYMAKER handles consistent and inconsistent data with the Dempster-Shafer rules of evidence combination, as given in (Shafer 1976). When a piece of evidence is provided the system, PLAYMAKER uses that evidence to support a specific value or set of values for an attribute. For example, if the user expresses high belief that the property (primary geometry) has the values horizontal and parallel, then both basin and shelf form a subset of values for the attribute depositional setting. When conflicting evidence is provided, the belief in any one value or subset of values for an attribute will be low.

PLAYMAKER is built upon an expert system shell known as MIDST (Mixed Inferencing Dempster-Shafer Tool) (Bach 1986). MIDST allows both forward- and backward-chaining. PLAYMAKER is partitioned in a manner similar to the major partitions of the attribute matrix as indicated by Figure 2. As a first step, it
attempts to determine the availability and quality of the various types of data, e.g., seismic, well, and field data. If, for example, no seismic data are available, no question related to seismic information will be asked. By determining the quality of a data source the system knows how much belief to attach to responses for which that data source is used.

Responses to questions are of two types. One type of question requires a "yes/no/don't know" response; the other requires that the user indicate a belief in the presence or absence of a property and also specify the quality of the data source (Figure 3). For the question about depositional setting, the user indicates one of five levels of belief that the play has that property. These are interpreted as 0.2, 0.4, 0.6, 0.8, 1.0, respectively. No response for a property is interpreted as nil belief in that property.

At run time PLAYMAKER presents "blocks" of related questions to the user. The texts of the questions are presented on the screen without responses. This is to give the user a chance to read several questions at once. A user can choose to respond to one of the questions by selecting it with the mouse pointer. At that time, the other questions being displayed move aside to make room for the selected question and its responses. The user can respond to that question and then move on to another in the same block. Questions in the same block may be selected and re-answered in light of answers to other questions if the user so chooses. The user must indicate with a click of the mouse with the pointer in the appropriate location that the block of questions is completed before the system will move on to another block.

PLAYMAKER also allows a corporate philosophy to be enforced. If plays are characterized uniformly according to rules derived from one or more company experts, district offices must follow the corporate approach to oil exploration, rather than develop individual approaches. Exploration strategies change, albeit rather slowly. When they do change, those changes can occur throughout the company by rewriting PLAYMAKER's rules and then circulating the new version of PLAYMAKER to all field offices.

Primary knowledge engineering was done in many interactive sessions between the mentor and computer scientists; this activity is described in detail in (Morgan et al. 1988). Further rules and relationships are derived by PLAYMAKER, which reasons about the type and quality of data available for each field or play. The rule base and inference structure for each captured field is built using MIDST (Mixed Initiative Dempster-Shafer Theory), which is a custom-built shell that derives a sequence of questions for each field that depends upon interactive user responses. PLAYMAKER has a rules editor that enables each user to customize the system to reflect proprietary exploration rules and corporate strategies. The current rule base has about 2500 rules concerning exploration keys. Version 1.0 of PLAYMAKER will be ready for b-release in the spring of 1990. A prototype of PLAYMAKER version 2.0 has also been built, and is now being tested and further extended (Biswas et al. 1990).

Report

REPORT enables the analyst to prepare a report describing characteristics of the target play and its analogs. The system will examine the attributes of the target play, and, based upon the values of the attributes found in (A), will prepare a description of the play. Some of the report will be driven by values found in (A). The analyst needs, of course, to write certain parts of the report, so REPORT provides the standard editing functions of a word processor. Construction of a baseline report which will serve as a template for every column of (A) is underway. Then, when the system is instructed to generate a report about a particular column of (A), the system will examine the values for this field or play, and refill slots in the template as needed. REPORT is written as a standalone LISP program, and will have a limited capability to learn how a particular company wants its exploration reports configured from examples. Each learning session will result in a new, permanent report template, which will then generate "uniform" reports across all columns in (A).
Sedpak

Greater understanding of a play can arise from an understanding of the sedimentary history of the basin in which the play is located. SEDPAK provides an interactive graphics-based forward simulation of the sedimentary processes by which the basin achieved its current sediment geometry and composition. An initial basin configuration is provided to the simulation, along with rates of accumulation for carbonates and the influx of sands and shales. The rise and fall of relative sea level is another major parameter. The intent of the simulation is that by matching the geometry of the new play as closely as possible to the output of the simulation, the user can better determine the geologic processes which generated current sediment geometry of the play and thus its potential for oil production.

SEDPAK provides users with a means for interactive analysis of seismic interpretations; users may input various control parameters that regulate simulated deposition over a specified scale of geologic time. The major variables of SEDPAK are eustatic sea level, tectonic movement, and deposition rates (clastic and carbonate). MOBIL, AMOCO, and TEXACO have been testing a beta release of version 1.0 of SEDPAK for about one year, and report very good correlation between simulated lithofacies and expert interpretations of seismic information. The current version of SEDPAK is a two-dimensional model that simulates deposition along the transverse axis of a basin; plans are underway to generalize this to a three dimensional simulation. Version 2.0 of SEDPAK will be ready for release in June of 1990, and will include listric faulting as a major upgrade in simulation capabilities. SEDPAK is the most well developed module of XX; interested readers may consult (Strobel et al. 1989; Kendall et al. 1989) for a detailed description of this part of the system.

Cluspak

CLUSPAK is the module of XX that endows it with one of its major capabilities: assistance to the explorationist in reasoning by analogy. Exploration geologists reason about potential hydrocarbon plays and prospects by analogy with past successes (and failures!), and their expert experience. XX attempts to capture as much of this facility as possible in the CLUSPAK module, which provides users with a means for exploring various clustering tendencies which may reside in (subsets of) columns of the attribute matrix. CLUSPAK will utilize both conceptual and numerical clustering techniques to find known fields and plays in the resident database that possess properties similar to the target play or prospect. Queries about analogs to target plays can be constrained in various ways, such as, but not limited to, analogs over: a single basin; a geographical region (e.g., the Gulf of Mexico); fields that possess similar properties over some subset of attributes; and so on.

SUMMARY

The XX system may be viewed as a large, intelligent database system that performs three essential functions. First, it enables companies to represent, through customization of PLAYMAKER rules, their data in a compact manner which is efficiently organized for exploration and retrieval. Secondly, XX provides explorationists with a tool which builds analogs to target plays, thus facilitating reasoning by analogy about the oil production potential of the prospect under study. Experts can determine the properties and relevance of data sources in determining analogs. Incomplete, inconsistent, and ambiguous data are handled in a uniformly consistent manner. And finally, XX enables a company to capture, store and enforce corporate experience and philosophy by embedding this knowledge in the rule base.

REFERENCES


237


ACKNOWLEDGMENT

This research was supported by grants from the MOBIL, BP, AMOCO, JNOC, TEXACO and UNION Oil Companies.
Figure 2. Partial Hierarchical Structure OF Attribute Matrix (A)

Minor Keys
(Attributes)

Values (v) e(v)
Limestone 0.00
Sandstone 0.25
Conglomerates 0.50
Shale 0.05
Gypsum 0.00
Anhydrite 0.05
Salt 0.00
Coal 0.10
Pyroclastic 0.00
\[ \sum e(v) \leq 1 \]

Major Key
Reservoir Character

Figure 3. Rule Editor in PLAYMAKER

<table>
<thead>
<tr>
<th>LHS</th>
<th>Value</th>
<th>Rank</th>
<th>Attribute</th>
<th>Value 1</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use.Well</td>
<td>Yes</td>
<td>3</td>
<td>Car.Eva.Fac</td>
<td>Buildup</td>
<td>Barrier</td>
</tr>
<tr>
<td>Dep.Set</td>
<td>Shelf</td>
<td>2</td>
<td>Car.Eva.Fac</td>
<td>Shelf-up</td>
<td></td>
</tr>
<tr>
<td>Fauna</td>
<td>Diverse</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Belief in Evidence 8

Quit Add Clear Delete Change Get Rule 1208

<table>
<thead>
<tr>
<th>Attribute-Value Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dep.Set - Shelf</td>
</tr>
<tr>
<td>Fauna - Diverse</td>
</tr>
<tr>
<td>.</td>
</tr>
</tbody>
</table>

239
TOWARDS A MORE GENERAL ARCHITECTURE FOR INTELLIGENT TUTORING SYSTEMS

Harald W. Kegelmann*, Lois W. Hawkes*, Sharon J. Derry**

Department of Computer Science*
Department of Psychology**
Florida State University
Tallahassee, Florida 32306

ABSTRACT

Most of the existing training and tutoring systems are individually hand-crafted applications that have little or no use in other domains. In this paper we describe a general control and knowledge representation architecture for Intelligent Tutoring Systems (ITS) that has a high degree of portability. Our approach is based on a hybrid representation of knowledge integrating object-oriented concepts with a fuzzy temporal relational database and constraint propagation.


1. INTRODUCTION

Today's technological society requires a highly trained work force in order to compete successfully in an increasingly global market. The accelerating pace of technological development brings on the need for periodic retraining of this work force [30]. This training includes the continuing education of staff, the training of new recruits as well as the retraining of personnel for new positions. In 1987, West-German companies spent more than $14 billion on further education for their already highly qualified employees. Further education is considered as an indispensable investment in the creative power and know-how of employees. It is supposed to enable the work force to cope with constantly changing demands at the work place, improve promotion, avoid unemployment and last but not least maintain and enforce the competitiveness of the West-German economy [19].

The crisis in education puts retraining of the work force in the U.S. at a much more elementary level. It is estimated that industry may soon be forced to hire a million workers who cannot read, write, or do elementary arithmetic. In Burlington, Vermont, for example, IBM is teaching its entire work force of 8,000 people Algebra I and II [41]. The cost to industry of providing this kind of training, including the loss of productivity, is estimated at $25 billion per year [23].

Sophisticated tutoring and training systems could address this situation. Tutoring systems combine both teaching and tutoring: the use of a structured lesson plan which has high level goals, the flexibility for attending to individual needs, and immediate feedback. Some researchers even predict that they can "enable us to carry out a revolution in education" [24, p. 204]. Good training software, on the other hand, can lower training costs and shorten the time required for personnel to become proficient in a new skill.

Despite this promising outlook, most of today's ITS are still confined to the realm of the research laboratory. In order for them to be commercially successful, they have to become more flexible such that the development of an ITS can be accomplished much more rapidly. One way to achieve this is through more general architectures [5], [21], or the development of ITS shells [25]. Ohlson points out that: "What we need, then, are not particular, quickly outdated, computer tutors but the know how of tutor construction" [22, p. 294]. We need to find principles that can guide us in the construction of ITSs. Research in that area would lead to an environment that provides tools for rapid prototyping and development of tutoring systems.

We are developing such a set of tools. In particular the TAPS system (Training Arithmetic Problem-Solving Skills) currently being developed is a step in that direction: it is designed to improve the arithmetic and problem-solving skills of 4th to 7th graders by teaching them how to solve word problems [9], [10], and is also applicable up to remedial college level by varying the vocabulary. This domain serves as a testbed for research of both general architecture, and development tools for problem-solving ITSs. To achieve this, we incorporate a hybrid representation and identify the structures and activities in the
system considering what components interact with each other upon what information, independent from the domain. This approach is guided by the view that the individual components of the system are active, interacting experts in the realm of diagnosis and tutoring.

2. CRITICISM OF CURRENT SYSTEMS

An evaluation of current ITS implementations reveals several problem areas in ITS system designs:

- **Modularity:** Most current systems are complex, unwieldy, and lack modularity [5]. To help developers maintain control and understanding of large, complex systems, knowledge should be organized and displayed in a clear modular format.

- **Flexibility and extensibility:** Most systems are limited to a particular domain or problem-solving methodology and often do not allow the integration of new domain knowledge or new pedagogical approaches.

- **Reusability:** Each new system is built from scratch. It is rare that one research group uses the control and knowledge representation schemes of another group [21]. Moreover, the representation schemes are not general or flexible enough to become a standard or tools for ITS development.

- **Pedagogical Foundation:** Many systems have been built which incorporate little understanding of how successful human tutors teach. They are mostly designed from the intuitions of AI researchers, often without the collaboration of an expert teacher working in that domain [21].

- **Instructional Theory:** The implementation of an instructional theory is highly experimental in nature. Researchers face a “lack of a general theory of instruction” [16, p. 178]. Although certain areas in human learning have been usefully investigated, much remains a matter of speculation.

- **Transparency:** Despite the complexity, the system must still be transparent for psychologists and subject-matter experts involved in the project. Revising theories should not require a substantial computer science background, rather the system should make viewing and updating of student models and tutoring rules fairly easy.

- **Portability:** None of the research literature discusses the issue of portability. ITS systems, in general, are designed around the only hardware dependent component: the interface. However, in order to become commercially successful, ITS systems need to run on different hardwares.

A thorough design of the system architecture has to take these aspects into consideration. The proper selection of a knowledge representation formalism is therefore crucial for the success or failure of a system.

3. KNOWLEDGE REPRESENTATION

In TAPS we opt for a hybrid representation of knowledge that centers around an object-oriented representation, incorporating fuzzy pattern matching equations, a fuzzy temporal relational database, and constraint propagation.

Hybrid representations have received much attention in recent years. Sloman in [26] argues convincingly that different representational formalisms are needed for the adequate representation of knowledge.

3.1. OBJECT-ORIENTED REPRESENTATION

3.1.1. Advantages

Object-oriented systems are characterized by the concepts of data abstraction and information hiding [28], [20]. Together with mechanisms for establishing hierarchical relationships, and inheritance of information, they provide a means to organize large software systems in a modular, well-structured way consisting of reusable code components.

Object-oriented concepts for the representation of domain knowledge in ITS have become popular recently. In particular, the Learning Research and Development Center at the University of Pittsburgh implemented several object-based systems [8], [18]. Murray in [21] applies this representation paradigm to the design of a general architecture for ITS. However, these systems differ from TAPS in that they use just one representation formalism.

3.1.2. The Expert System Shell FORK

The ‘core’ of the tutoring system TAPS is represented in the expert system shell FORK [9], [4], [15]. FORK is implemented in CommonLISP [15] and is available on various hardware platforms. Its kernel is completely object-oriented but has been extended by a forward-chaining, data-driven production system, a blackboard system similar to BBt [13] and a truth maintenance system.

In its present form, the rule-oriented component of FORK is more powerful than that of LOOPS [9], because it “offers additional means for conflict resolution and for processing vague information” [4].

The programming environment of FORK is ideal for system development: it provides a means for editing and handling objects; a browser that supports viewing and manipulating of object hierarchies; and sophisticated debugging tools that allow tracing of both the communication between objects and the execution of rules and methods.

3.2. FUZZY PATTERN MATCHING EQUATIONS

As opposed to classic set theory, where an item either
belongs to a set or does not belong to that set at all, fuzzy sets have a "degree of membership". As an analogy to tests, true-false are classical, and essay are fuzzy. It could be that the answer is not exactly correct, but also not exactly wrong. Assessing partial credit is an assessment of a fuzzy value. This is an important concept in capturing imprecision and vagueness inherent in real world data.

A technique has been developed for incorporating such imprecisions in the system's evaluation of the student, his capabilities and identifying characteristics. These "fuzzy" or imprecise variables are part of the student model (see 4.3.) and are used in fuzzy pattern matching equations to determine the student's current level of expertise, motivation, position in the lesson plan, etc.

3.3. CONSTRAINT PROPAGATION

These variables are not independent and the same variable can appear in various equations. The modification of one of the variables consequently requires reevaluation of all equations that contain this variable. One would like to have a system such that once a variable is updated, the new value would automatically propagate throughout the collection of equations. We believe that a constraint network as described in [27] is a viable approach.

In our object-oriented implementation, the components (i.e. equations) are represented by objects which are connected with each other by explicit pointers. In propagating values, the visited components are reevaluated. There is a possibility that cycles could arise in this network and cause it to fall into a state of self-resonance. We are currently investigating this possibility and considering a modification of the algorithms suggested by DeKleer and Williams [8] and Fendler [12] for truth maintenance systems.

4. SYSTEM COMPONENTS

In TAPS we distinguish between four different system components as it is common in most ITS systems. Yet, we consider the individual subcomponents as experts in their own right that "know" what to do if certain events take place and with whom they have to communicate to address the situation.

4.1. THE COMMUNICATION MODULE

The design of current ITS is closely tied to how the domain is presented to a student, that is, the tutor's interface. However, the interface in most systems, once it is implemented, is static and provides no user-sensitivity. If we accept Wenger's suggestion that the goal of tutoring systems is to communicate knowledge [29], we also should accept that there is not a single best way to communicate any kind of knowledge. It has been suggested that the user interface can make the difference between software that is useful and software that gets used. Moreover, the interaction between an ITS and the way it presents itself is even more important in determining success or failure than for a regular application interface.

As a consequence, the design and construction of the Communication Module (CM) in TAPS was driven by some of the principles and guidelines for user interface management systems described in [7], [17] and [2]:

- Separating the functions of an interactive application from its interface. This is essential for reusing interface components and simplifies their modification and extension. It should be expandable in order to enrich the way knowledge is communicated.

- The interface should be dynamic, i.e. it behaves like an intelligent observer who automatically adapts the interface to the habits, expertise and preference of the user. To accomplish this, we incorporate a knowledge base in the control unit of the CM [11]. This is a step beyond Ohlsson's notion that effective ITS "must distinguish between the subject matter and the formats in which it can be presented" [22, p. 217].

- To enhance the independence between what the system presents and how it is presented, the CM processes the flow of communication in and out of the system via an internal data interface (IDIF).

- Portability, an issue neglected in many systems, ensures that we can move the system quickly to new hardware environments. We enhance portability by using the industry standard CommonLISP for implementation; hiding the internal workings of the interface with the high level representation of the interface; and developing a basic set of machine-independent operations that map on hardware specific graphic functions.

4.2. THE EXPERT SOLUTIONS MODULE

The Expert Solutions Module (ESM) consists of a collection of objects that interact with the Tutoring Module (TM, see 4.1). We distinguish between objects such as: error categories, knowledge components, and schema representation.

Abstract error categories that have been obtained from think-aloud protocols capture the criticality and causes of the error category, the instructional goals that need to be set for remedy and possible tutorial intervention.

Knowledge components are individual data items that consist of achievement scores, learning ability score etc. for the knowledge that is to be learned. They are ordered in a hierarchical network structure denoting prerequisite learning relationships. Fuzzy values for these data represent the strength or weakness of a student's performance levels or mastery of a specific knowledge component. Fuzzy pattern matching is used to compare the student's current performance behavior and performance history to "known" performance patterns. This comparison is then used to determine "how closely" the student's performance behavior matches that of a particular type of problem solver. Once a student's behavior has been
identified with a particular type of problem solver, the characteristics of that problem-solver type may then be used to assist in tutoring and to update that student's individual achievement levels and learning goals [14].

The problem representation in the ESM is based on the identification of sets and their interrelationships in word problems. The sets are grouped into one of four different types of relational schemas based on the relations that hold between the sets, i.e., combine, change, compare, and vary schema types. Sets and schemas can be grouped to form more complex problems, building a collection of binary trees that reflect all semantically meaningful solution paths of the problem.

Non-schemas, i.e., schemas that can be constructed from the description of the problem, but contain sets that are unrelated in the problem and therefore do not match a problem schema, are represented as well to record if the student combines irrelevant information.

Problems will be constructed by the use of templates and user-personalized dictionaries [14] containing information about the student's age, world knowledge, gender, and interests.

4.3. THE STUDENT MODEL

TAPS provides both an offline and online student model. The mechanism which is used to store the student knowledge model is referred to as an Intelligent Fuzzy Temporal Relational Database (IFTReD) [14] and contains the history of the student and his past performances. Advantages of this representation are: its flexibility, its efficient means for updating the database and its acceptance of fuzzy and temporal information. We obtain this information by taking "snapshots" of student performances at different time intervals to establish a performance pattern over time.

There are currently seven categories of student information stored in the database, most of them are not domain-dependent: basic schema knowledge, the ability to solve problems of various degrees of difficulties and complexity, strategy and meta-cognitive knowledge, motivational variables, error and performance history, preference information and personal data.

The Dynamic Student Model (DSM) receives initial data from the database about the students' past performance and strategies on the selected problems and instructional goal. Trace information, i.e., the activities of both the student and the system, knowledge about his performance on the current problem category are recorded and updated online during the session.

The IFTReD operates primarily offline, using information from the DSM to update its tables. Information will be accessed in two ways: by the teacher/researcher using a query language and by the TM using various access functions.

4.4. THE TUTORING MODULE

Much like the other components of TAPS, the Tutoring Module is a collection of active objects. To gain domain-independence we separate the instructional goals and knowledge components, i.e., what we tutor, from the tutoring rules and strategies, i.e., how we tutor. Tutoring in the TAPS system is plan-based and opportunistic [9]. Plan-based tutoring is accomplished by the use of global instructional and achievement goals and the system's Lesson Planner, whereas local diagnosis is associated with the opportunistic context and performed by the Diagnoser.

The Lesson Planner (LP) works primarily offline and establishes an instructional goal for the next tutoring session. Once the instructional goal has been established the LP selects from various problem types and instructional routines in order to design a tutorial session which will accomplish the instructional goal. If the student experiences difficulties in solving the problems of the current session, the LP will perform an online revision of the lesson plan.

The Diagnoser receives data from the CM and matches the students' solution path and his problem-solving strategies with the tree representation in the ESM to identify possible errors. Deviations from the student's hypothesized strategy are reported to the Mentor and the corresponding error category is consulted for more error specific information.

The Mentor object consists of several subclasses that reflect the different modes of the Mentor such as coaching, reactive and mixed initiative. Tutoring rules associated with these modes contain fairly general, domain-independent knowledge about how to teach. The Mentor also maintains a trace of the session that is used online to assess different interventions and offline to update the database.

Although TAPS addressed the domain of arithmetic word problems, we think that the concepts of tutoring and diagnosis employed are generalizable to any domain.

5. CONTROL ARCHITECTURE

The control structure is similar to the one proposed by Murray [21, p. 50], separating between action-, control-, and knowledge-subsystems. The CM represents the action-subsystem, containing presentation objects, a knowledge base, and a control unit that determines upon the student's preferences stored in the knowledge base what presentation objects to activate. It also controls the flow of information between the CM and the control-subsystem.

The control-subsystem consists of the Lesson Planner, the Mentor and the Diagnoser. They assist each other in the evaluation of the student's performance, assessing error categories, intervention of the problem solving process and, if necessary, revising the lesson plan. In order to perform these tasks they access and update information from the ESM, the database and the Dynamic Student Model which form the knowledge-subsystem.
6. CONCLUSION

We presented a general architecture for problem-solving ITs that achieves its reusability and portability by: decomposing the system into a collection of communicating experts; the extraction of domain independent structures and functionality; and the abstraction from hardware-dependent graphics. A subset of the proposed system has been implemented on a XEROX 1186 which is used as the environment to develop the tutoring aspects of TAPS. The system is currently being reimplemented on the NeXT machine, which offers faster execution and an Interface Builder (IB), a powerful tool to develop interfaces. The IB should make it fairly easy to implement and test the acceptability and effectiveness of various interface configurations to provide an effective tutoring and training environment.

References

An Architecture for Rule-Based Knowledge Representation and Parallel Inferencing

Dan E. Tamir, Robert A. Morris, Mordechay Schnieder

Department of Computer Science
Florida Institute of Technology
150 W University Blvd.
Melbourne, Florida 32901

Keywords:

Abstract
A bit-matrix can serve as the ultimate data structure for knowledge representation and inference in a rule-based expert system. Using this representation, the process of inference is implemented by matrix manipulation operations. This approach leads to a simple architecture for knowledge management. It enables fast execution of the expert system and the implementation of parallel processing techniques in the process of inference.

Introduction
In this paper we demonstrate that a bit-matrix can serve as the ultimate data structure for knowledge representation and inference in a rule-based expert system. After a short cycle of pre-processing the entire expert system (both facts and rules) is represented in a bit-matrix. The process of inference is then completed by matrix manipulation operations. This approach leads to a simple architecture for knowledge management. It enables fast execution of the expert system and implementation of parallel processing techniques in the process of inference. Moreover, the proposed approach enables hardware implementation of the expert system (e.g. on a wafer scale chip). Hence, expert systems implementing this approach can be used to support critical applications that require fast response and real time performance.

In the general form a rule-based expert system is a computer implementation of a formal system. The axioms are given in the form of atomic formulas (facts) and implication formulas (rules). The inference rule is modus ponens. An inference engine applies a process of chaining facts and rules (forward or backward) to derive conclusions that logically follow from the axioms. Generally, the axioms are clauses representing formulas from first order logic and may contain variables. However to simplify the discussion we first consider constant formulas (formulas with no variables). We will later show how the proposed architecture can be integrated into a logic programming environment where the formulas may have variables.

As an example consider a rule-based expert system containing the following set of axioms (facts and rules):

A1: A.
A2: B.
A3: A → C. /* If A then C */
A4: B, C → D /* If B and C then D */

The inference engine of the expert system can apply modus ponens on axioms one and three to derive the conclusion A5: C and then by modus ponens on axioms two and four using A5, the system can derive A6: D.

A variety of data structures are used to store facts and conclusions in existing expert systems. Traditionally a database is used to store facts and derived conclusion. Other data structures are used to represent rules. Some of them are lists, stacks, queues, and matrices.

Many expert systems use lists to represent rules. The inference in these cases involves pattern matching. Pattern matching consumes a major part of the time spent through the process of reasoning in the expert system and becomes the bottleneck in many cases (Moldovan 1989). Schneider has implemented an expert system shell (MAD) where a matrix is used to represent
the inter-dependency between rules (Schneider 1988). This increases the speed of the inference engine. Tamir has shown that a dependency matrix takes advantage of the transitivity relation between rules, hence, it can increase the speed of chaining rules (Tamir 1989). Actually the transitivity matrix is a compact representation of the derivation tree of the expert system. In many cases a decision process can be represented by a decision tree. The transitivity bit-matrix is then the adjacency matrix of that tree. Tamir develops a unique approach, where every rule is a C function, and the transitivity matrix is a matrix of pointers to these functions (Tamir 1989a). Under this schema the process of chaining resembles a nuclear reaction where a "fired" rule automatically activates all the rules that contain its conclusion as one of their premises. Moldovan (1989) investigates issues of performance in expert systems and reports on a multiprocessor for an expert system. In his system a bit-matrix is used to represent rules. This matrix is used to reduce the search space for pattern matching.

A bit – matrix representation of the entire expert system

The systems reported above use transitivity matrices as a data structure for the representation of rule interdependency and achieve improvements in the execution time. However, facts are still represented in a different data structure (database). In this paper we propose a new approach where both facts and rules are represented in a bit-matrix. This extension makes the expert system regular and enables using the bit-matrix as the ultimate data structure in the system.

In order to explain the way we construct the bit-matrix, let us reconsider the example given above. Using the method described by Moldovan (1989), the resultant system consists of two facts F1 and F2 stored in a database, and two implication rules I1 and I2 represented in a bit-matrix M (note that F1=A1, F2=A2, I1=A3, I2=A4). In general, M is a k by k matrix (where k is the number of rules in the system). The entry M_{i,j} is set to 1 if one of the premises of rule R_i is the conclusion of rule R_j. Otherwise M_{i,j} is set to 0.

To represent facts in the same bit-matrix with rules we rewrite facts as rules. Let \[ \square \] denote the valid formula (formula that is true under any given interpretation). If it is known that A is true then following from the logical definition of implication, the fact A is logically equivalent to the rule \[ \square \rightarrow A \]. Hence the axioms A1-A4 given above can be rewritten as the rules:

R1: \[ \square \rightarrow A \].
R2: \[ \square \rightarrow B \].
R3: \[ A \rightarrow C \].
R4: \[ B, C \rightarrow D \].

Using Moldovan approach (Moldovan 1989), the resultant bit matrix is:

(R1) \( (0 0 0 0) \)
(R2) \( (0 0 0 0) \)
(R3) \( (1 0 0 0) \)
(R4) \( (0 1 1 0) \)

The reader may be concerned that actually we increase the dimensionality of the matrix by adding row of zeroes for every fact. Although this approach increases the dimensionality of the bit-matrix it has the advantage of unifying the treatment of facts and rules. The entire expert system is represented in a bit-matrix. Moreover, since the entire process of inference can be carried out on the matrix using relatively simple operations, it is possible to parallelize the process. Furthermore, special hardware for matrix manipulation (e.g. array processors or special purposes chips) can be used to accelerate the reasoning process.

The process of inference can be performed using the bit-matrix with almost no reference to any other data about the expert system. To demonstrate this, we consider two operations on a matrix. The first is obtaining the n-th power of a matrix under “max, min” operations. The second is an operation of “zeroing” elements in the matrix.

**Definition 1:** Let \( A = [a_{i,j}], B = [b_{i,j}] \) be two bit-matrices of order \( N \times N \). Let \( x + y \) denote \( \max\{x, y\} \), \( x \cdot y \) denote \( \min\{x, y\} \), and

\[
\sum_{i=1}^{n} x_i = \max\{x_1, x_2, ..., x_n\}.
\]

The matrix \( C = A \times B = [c_{i,j}] \) is obtained from the matrices \( A \) and \( B \) by:

\[
c_{i,j} = \sum_{k=1}^{N} a_{i,k} \cdot b_{k,j}.
\]

Data base:
F1: A.
F2: B.

Bit-matrix:
(I1) \( (0 0) \)
(I2) \( (1 0) \)

246
Definition 2: Let $M$ be a matrix we define the $n$th power of $M$ ($M^n$) recursively:

1) $M^1 = M$
2) $M^i = M^{i-1} \times M$ \hspace{1em} ($i > 1$).

If a row in $M$ contains only zeroes it is referred to here as a zero row.

Definition 3: Let $M = |m_{i,j}|$ be a bit-matrix so that $k$th row of $M$ is a zero row and the row $n$ contains '1' at the $k$th column. Then the matrix $M'$ obtained from $M$ by setting $m_{k,n}$ to zero is called the backward matrix of $M$.

Theorem 1: Let $M$ be the transitivity bit-matrix of an expert system $E$ and let $M^i$ denote the $i$th power of $M$. If the $k$th row of $M^i$ is a zero row then the conclusion of the rule $R_k$ is a logical consequence of the axioms of the system.

Theorem 2: Let $E$ be an expert system and suppose that the rule $R_k$ in $E$ is: $r_1, r_2, \ldots, r_{n-1} \rightarrow r_n$ \hspace{1em} ($0 \leq k \leq n - 1$). Let $M$ be the transitivity bit-matrix of $E$ and let $Z$ be the backward matrix of $M$ with respect to $m_{i,k}$. The atom $r_k$ is a logical consequence of the axioms of $E$ if and only if $m_{i,k} = 1$ and $z_{i,k} = 0$.

Theorem 3: Let $E$ be an expert system, let $M$ be the transitivity bit-matrix of $E$, let $n$ be the order of $M$, and let $M^n$ be the $n$th power of $M$. The $i$th row of $M$ is a zero row if and only if the conclusion of rule $R_i$ is a logical consequence of the axioms of $E$.

We say that a matrix $M$ has no backward matrix $M'$ if the operation defined in definition three can not be applied on $M$.

Theorem 4: Let $E$ be an expert system and $M$ the transitivity bit matrix of $E$. $M$ has no backward matrix if and only if for every zero row $i$ the conclusion of the rule $R_i$ is a logical consequence of the axioms of $E$.

The proof of these theorems is omitted from this paper.

The practical result of theorems one to four is that the inference in the expert system can be implemented by any sequence of multiplication and zeroing performed on the bit-matrix of the system. The process terminates in one of the following cases:

i) $M'$ is a zero matrix
ii) $M$ does not have a backward matrix
iii) $M = M^i = 1$ for some $i$
iv) $M' = M^n$ have been obtained

The selection of a sequence depend on considerations of efficiency and varies in different systems and applications. Multiplication has the advantage that it can be used to detect circularities in the expert system. Zeroing have the advantage that it can be implemented simply by shift registers. Hence a hardware realization of a system implementing the zeroing operation may be economic. On the other hand multiplication under 'max', 'min' is simpler then regular multiplication and can be implemented by half adders.

A system for fault isolation

We have implemented an expert system for fault isolation. The system assists in detecting mal functioning components of a tested electronics board (Eliot 90). This system utilizes the principles discussed above. In the system the knowledge base is extended to include facts. A fact $F$ is represented as the rule $\square \rightarrow F$. The knowledge base is associated with a bit-matrix ($M$). $M$ is a $N$ by $N$ matrix where $N$ represents the number of rules in the extended knowledge base. $M$ is constructed according to the example given above. The bit matrix is created when the expert system is invoked and is updated when a new rule is inserted into the knowledge base or a rule is deleted from it.

The bit matrix has 3 functions in our system:

1. It is used by a Question - Answer Program (QAP) to trace the inference path when describing how a conclusion was reached.
2. It helps the inference engine to guide the user to reach a conclusion when the data which was provided by the user is insufficient.
3. As described above, $M$ is used in the inference process to detect all the conclusions that follow from the axioms in the system.
4. The bit matrix assists in validating the system. If the matrix $M'$ ($i = 1, 2, \ldots, N$) has an entry '1' in the diagonal then the expert system contains circularities.
Traditionally the inference engine of an expert system screens all the rules. In the fault isolation system the inference is performed by bit-matrix manipulation. Currently the matrix operations are implemented by software. This raises an issue of performance analysis. Bouni compares the performance of several versions of expert system shells derived from MAD (Bouni 1989, Schneider 1988). He examines a version written in PASCAL, with a version written in Ada. He also conducts a performance analysis on the influence of different coding schemes for the rules of the system. He shows that a compiled version of the rules outperformed a traditional string level screening of the rules. A compiled version is analogous to a bit-matrix representation and restricting the matrix manipulation operation to the zeroing operation as defined in definition three. This supports our claim that using a bit-matrix representation and manipulations (even in software) will improve the real time performance of an expert system.

The hardware implementation of a matrix-based inference engine, or using special architectures (e.g., array processor) for the matrix manipulations is expected to increase in a significant way the speed of execution. We are currently performing preliminary research aimed at implementing the bit-matrix manipulation using a special hardware on a wafer scale chip. We expect this implementation to lead to a fast expert system suitable for real time applications.

Integrating the bit-matrix with logic programming languages

So far we considered constant formulas. In an expert system that includes dynamic variable binding, reasoning is generally performed in one of two ways:
1) Considering a related set of constant formulas in the Herbrand universe (Chang 1973).
2) Using unification and resolution as in PROLOG (Delahaye 1987).

In the first approach the reasoning is performed in steps. In each step a set of constant formulas is constructed (generally by instantiation) and used to deduce new conclusions. A bit-matrix representation can be integrated with this approach. The bit-matrix is used to perform the deductions. The integration can be implemented using the following outline:

At each step do:
   i) Instantiation
   ii) Construction of a bit-matrix
   iii) Deduction using the matrix manipulation operations defined in definitions one, two and three.

In the unification and resolution approach, each stage in the deduction procedure is performed on a set of (unified) first order formulas. Syntactically, in each stage, the set of formulas can be treated as constant formulas since no variable binding occurs until the next stage of unification. Using the bit-matrix in conjunction with a unification procedure can impose changes in the control strategy of the system and may lead to using modulo rather than resolution. A possible outline for integrating a bit-matrix with the unification procedure is as follows:

At each step do:
   i) Unification
   ii) Construction of a bit-matrix
   iii) Deduction using the matrix manipulation operations defined above.

Note that in this case the unification can be performed simultaneously on several pairs of clauses while in PROLOG each unification and resolution involves a single pair of clauses.

Summary

In this paper we have defined a bit-matrix representation of rule-based expert systems. Reasoning with such a structure consists of elementary matrix operations which admit of hardware realization and the use of parallelism to improve its real-time potential. At the same time the matrix can be integrated into a logic-programming environment with a loss of expressive power.
REFERENCES


A KNOWLEDGE-BASED SYSTEM FOR TRANSPORTING SOFTWARE

Soklei Leong and Fred Y. Wu
Department of Electrical and Computer Engineering
University of Miami
P.O. Box 248294, Coral Gables, Fl 33124

ABSTRACT

A traditional way to handle the problem of transporting software is to built translation software in an ad hoc manner, which is difficult to understand and reuse. In this paper, a knowledge-based system that combines AI and translation techniques is proposed to solve the software translation problem. In the proposed system two subsystems are included, an intelligent user interface for acquiring software translation knowledge and a meta system for constructing translation software. The resulting system is therefore capable of generating language-independent translation software based on given specifications, with both enhanced application flexibility and process automation.

INTRODUCTION

Using translation software is a traditional way to handle the problem of system incompatibility. Since translation software is not only system dependent but functionally dependent to a particular translation process, it is virtually impossible for a user to add or update the software. Therefore, in the case where a new utility program is to be adapted, new translation software will be required. Moreover, when a system is to be migrated to a new system, all its system dependent source programs need to be converted, which is a process with high risk of massive cost overruns or failure. Nevertheless, the need to convert software grows from year to year (Wolfberg 1983).

Recent research using transformational approaches can be divided into two classes: those are relatively limited in power but require no user guidance and those are capable of very complex implementations but only under user guidance (Rich and Waters 1987). Although improved solutions have been suggested and proved by those new approaches, the natural trade-off between flexibility and automation still seems to be an inevitable bottleneck.

A knowledge-based system is proposed to solve the above mentioned problems, in which two subsystems are included, an intelligent user interface for acquiring software transportation knowledge and a meta system for constructing translation software. The expected system is therefore capable of generating language-independent translation software based on given specifications, with both enhanced application flexibility and process automation.

FUNDAMENTAL PROBLEMS

The bottleneck of system incompatibility is due to the nature of traditional solutions by using translation software of which the translation scheme is implicitly implemented, neither understandable nor modifiable, for the translation software is usually written in a general purpose computer language (Leong and Wu 1989). One solution proposed by current most active research in automatic programming (Rich and Waters 1987) is based on the transformational approach by designing a super high level specification language to explicitly define each translation process as rules or specifications (Leong 1987; Freudenberger, Schwartz, and Sharir 1983), and therefore highly improved readability and flexibility of translation software is obtained. However, the problem of the semantic gap between specification languages and computer languages familiar to human experts may introduce difficulty or complexity in other aspects (Wu 1989), for instance, using or learning the specification language. Taking the advantage of expert system technology, another solution is an intelligent system which collects human expertise in program translation from a human expert through a knowledge engineer and then serves as the human expert in the subsequent software migration work (Rich and Water...
1988). This solution waives the problem of using specifica-
tion languages for human experts, however, the process of
knowledge acquisition has also proven hard due to an-
other semantic gap raised between human experts and
knowledge engineers; therefore the semantic problem has
eventually not been solved, but been unloaded from hu-
man experts and imposed on knowledge engineers.

The natural trade-off between flexibility and automa-
tion results in another bottleneck of the current research
in transporting software. None of the existing software
transporting system has achieved both enhanced flexibil-
ity in applications and full automation in software trans-
portation (Rich and Waters 1987).

METHODOLOGY

A knowledge-based system composed of an intelli-
gent user interface and a software translation subsystem
(Fig. 1) is proposed by using both AI techniques and
translation schemes. Knowledge about transporting soft-
ware is acquired through the intelligent user interface of
the system, which allows the human expert to directly
transfer required knowledge to the system without the
help of knowledge engineers and specification languages,
free from semantic problems mentioned above. Received
information is then automatically translated into an in-
ternal specification language (CRDL (Leong 1987)) which
is accepted as the input of the software translation sub-
system for generating translation software, and therefore
highly automated translation processes in flexible appli-
cations could be obtained.

Knowledge domain analysis

The rule-based representation is a dominant scheme
for encoding expert knowledge and has been widely adopted
in many rule-based systems designed specially for those
problem areas of which the domain knowledge can be
obtained through conversation. The knowledge of trans-
porting software, however, has been found hard to be
represented in rules; moreover, the knowledge itself is
not completely vocal expressible and transferable (Wu
1989). Therefore, a user interface is very important in
the whole design to provide human experts enough tools
in transferring their knowledge of program translation,
while a decent analysis of the specific knowledge domain
almost determines the design strategy and the resultant
performance of such an interface. According to the result
of the knowledge domain analysis, knowledge involved in
program translation is classified in two levels as follows.

A. Knowledge of source and destination languages

A general understanding of the grammars of both
the source and the destination languages and their dif-
ferences, which eventually determines the initial and the
goal states of the program translation problem, should be
first required to perform a lossless translation. Knowl-
edge primitives in this level include the understanding of
two identities:

1. Token specifications: The definition or declaration
   of each basic element (token) of the language.
2. Syntactical features: Descriptions of relationships
   between tokens in composing the language.

B. Knowledge of translation

Based on the understanding of grammars of both
the source and the destination languages, the knowledge
of program translation can be abstracted in the form
of mapping functions that transform each specific feature
from the source language to the destination language.
Knowledge involved in this level can be further classified
in three groups as follows:

1. Location identification: For each conversion pro-
cess, it is usually necessary to locate the element to
be modified in the source syntax tree as a starting
point for translation.
2. Condition specification: Unlike those that need be
defined as source-destination patterns at the syn-
tactical level, some conversions are not purely de-
termined by syntax; in this case, the attribute of
a syntactical element can be viewed as a condition
for the transformation.
3. Actions: After the location or condition being spec-
ified, specific actions to be taken should be provided
to describe the translation process.

Intelligent user interface

A user-friendly interface is under development ac-
cording to a two-phased learning strategy developed from
the results of the knowledge domain analysis. The entire
learning process is directed by the interface in a question-
and answering manner.

A. Grammatical information acquisition

The intelligent interface will first prompt the human
expert for grammatical information of the source and the
destination languages. Queries will be generated accord-
ing to each non-terminal symbol defined by the human
expert for its descendant decomposition until the entire
grammar is completely defined, ending up with terminal
symbols.

Based on the first-step learning, both grammars of
the source and the destination languages are understood
by the learning system, the former as given initial conditions, the latter as specified goal states. Conversions involving direct mapping between terminal and non-terminal symbols can be immediately determined at the end of the first-step learning cycle by comparing grammars of both languages. Semantic differences that can not be unified by mapping functions will invoke queries to prompt the human expert for translation procedures in the second learning phase.

B. Translation procedure acquisition

In this learning phase, the system will expect the human expert to input translation schemes for incompatible semantics. Each incompatible semantic feature of the source language as compared with the destination language is sequentially displayed, and human experts may use the system provided tools to describe the required translation processes. The tools include location identifiers, condition specifiers, and action procedures.

For location identification, the system provides human experts with location identifiers such as NEXT-TO, BEFORE, BETWEEN, and NTH-TERM for locating the element to be modified in the source syntax tree. For condition specification, the system provides human experts with condition specifiers, which are utility functions such as DATA-TYPE-CHECK and COLUMN-CHECK, to allow the human expert in specifying particular considerations for certain actions. The human expert can also define his own condition check procedures, by selecting the USER-DEFINED option. For action specification, the system provides human experts with action procedures, such as CONVERT-TO, OUTPUT, INSERT, and DELETE, for composing necessary procedures in converting the source pattern to the expected goal pattern. The human expert can also define his own procedures for necessary actions not being provided by the system.

Information received through the interface is organized by a frame-based representation (Wu 1989). Each frame contains information of a particular conversion process, or a defined lexical feature, and the built-in meta rules will instantiate the CRDL slot for each frame by converting the obtained specifications into CRDL forms which will be sent to the software translation subsystem for generating translation software (Fig 2).

Software translation subsystem

The software translation subsystem has two major tasks:

1. Translates source programs into target programs using source to target language translator.

2. Supports the construction of the above mentioned translator using a translator writing system.

The purpose for the second task is to reduce the effort of writing and maintaining a translator.

The key problem in developing a translator writing system is to establish a formal system which is oriented for describing the rules of translation from one language into another. That is, a language for describing the translation process is first designed. A compiler for that language can then be constructed to generate the code for the translator described. The translator can then be used to translate source programs into target programs.

A language, called Conversion Rule Description Language (CRDL), for translating programs from one dialect of a particular high level language into another has been developed (Leong 1987). In the CRDL, the conversion process is described in the most popularly used translation scheme, the Syntax Directed Translation (Aho and Ullman 1976). Besides features for describing syntactic characteristics of a language, CRDL also has features which are convenient for describing the conversion process.

A prototype compiler for CRDL has been implemented and has been proved to be quite effective for a skillful translator writer. By using the language the translator writer can concentrate on designing the translation mechanism without worrying about the underlying detail implementation. However, an interactive system which has the source level debugging facility has also been found to be important for the further success of the system.

CONCLUSIONS

Most of the work in solving the problem of transporting software has been mainly in writing translation software in an ad hoc manner. The resulting system built in such manner suffers from the problems of maintainability, adaptability and reusability. In this research, we attack the above problems by developing an intelligent translator writing support system using AI and translation techniques. A prototype of the proposed system is under development with the following technical merits.

Software Productivity The proposed system will interact with the expert who is responsible for writing a translation software at a very high level. The system will enforce the methodology for the specification that guides the expert to define a complete set of translation tasks. The expert can thus be relieved from identifying these tasks and other low level implementation details.

Maintainability and Adaptability Knowledge of translating software is explicitly represented as production rules. Therefore, it can be easily modified and applied to another translation.
Reusability The knowledge for well understood translation can be accumulated in the system knowledge base and reused.

Knowledge Transfer The proposed system can also be used to construct the inverter of a translator, which can be used by the users of the source language to learn the target language.

Learning Capability Software transportation knowledge is acquired based on a two-phased learning strategy through the intelligent user interface which provides human experts with a variety of tools to define knowledge.

The goal of this research is to provide not only a facility but also the technology for the automation of software transportation which will reduce the cost of transporting software significantly and help the industries in the following ways:

- Reducing the cost of transporting software from other machines for young computer companies.
- Increasing the availability of products on different environments for software companies.
- Introducing better technologies, for already established companies, which are not compatible to their existing systems in less awkward fashion.

ACKNOWLEDGEMENTS

This research is supported by Florida High Technology and Industry Council (FHTIC).

REFERENCES


Figure 1: Architecture of the Knowledge-based System

Figure 2: Translation Software Generation Process
HALO: A Fuzzy Programming Language

David F. Clark and Abraham Kandel
SUS Center for Artificial Intelligence and
The Department of Computer Science
The Florida State University
Tallahassee, FL 32306 USA

ABSTRACT. Borrowing from both high- and low-order languages, HALO (High And Low Order) integrates the symbolic programming and order-independent structure of expert systems with the control structures and procedural abstraction of Pascal. The HALO language is based on possibility theory and, consequently, is well-suited to expert system development. HALO, probably best described as a "fuzzy Pascal", is simple to learn and use yet is powerful enough for complex artificial intelligence applications requiring structure, modularity, process control, and uncertainty management.

INTRODUCTION

Artificial intelligence (AI) languages and expert system tools are designed to solve problems requiring some degree of intelligence. Although these languages are very well suited to solving certain classes of problems, many are so poorly structured that AI programmers have completely lost the step-by-step control found in algorithmic languages. LISP, the oldest of the AI languages, is not at all well structured and is therefore vulnerable to errors and breaches of privacy (Tennent 1981). Furthermore, LISP and many of the AI languages which followed are poorly structured and are not general-purpose (Burton 1987).

Neither the fifth generation languages such as PROLOG (Clocksin and Mellish 1982) and Pop-11 (Burton 1987) nor production systems like OPS5 (Brownston et al. 1985) and MAD (Schneider et al. 1989) offer a practical solution to this problem. The order-specific Horn clauses of PROLOG dictate some structure, but this is contrary to the spirit of true logic programming (Forsyth 1984). Furthermore, PROLOG has very little error protection, no scope whatsoever, and a gigantic built-in database. Production-rule systems do not provide a solution because they are not appropriate for structured problems (Brownston et al. 1985) and are restrictive to use (Burton 1987). Moreover, the modularity inherent in a rule-based system is limiting, particularly in an uncertain environment (Morawski 1989).

In developing a number of expert system applications at Florida State University (Schneider and Kandel 1988; Schneider et al. 1989; Clark and Kandel 1990a), the authors observed the utility of conceptually dividing AI applications into four parts: upper level meta-rules (rules defined on the store and the state of a program or knowledge base), middle level algorithmic control structures, lower level order-independent rules, and bottom level supporting code. Unfortunately, hybrid systems pose a number of problems not found in single-language systems.

The HALO language (High And Low Order) was designed primarily to synthesize high- and low-level modules into a single, structured environment. HALO provides a very high degree of abstraction for prototyping AI systems but at the same time facilitates structured programming and exploits the computer's low-level facilities. On the surface, HALO appears syntactically similar to LISP, but its semantics are actually closer to a "fuzzy Pascal". HALO, an experimental language implemented in ANSI C (Kernighan and Ritchie 1988), currently runs on MS-DOS and UNIX systems but may be ported to any system with an ANSI C compiler. Since HALO directly interfaces with external C and assembly language code, HALO programs may be simultaneously of enormously high- and low-order.

SYNTAX

A HALO program is a collection of F-expressions (short for fuzzy expressions) which are either symbols or lists. A list is enclosed in square brackets and, if nonempty, contains a symbol followed by a (possibly null) list of F-expressions. F-expressions are syntactically similar to the S-expression list notation in LISP but do not reduce to pairs. The HALO interpreter maps F-expressions into fuzzy values, often producing side effects such as declaring variables or writing to a stream.
The simplest F-expressions (literals, symbols, and procedures) appear without brackets. When evaluated, a literal returns itself, a symbol returns its value (provided that it is bound in the current environment), and a procedure displays its definition and returns a true value. When a list is evaluated, the first symbol in the list names the procedure to call and the remaining F-expressions serve as parameters for the call. When a procedure name is evaluated outside of a bracketed list, it displays its definition and point of entry.

DATA AND BINDING

The primary data type in HALO is the fuzzy quantity which is represented by a real number. The limits of the fuzzy interval and the threshold of truth are user-definable and default to 0.1 and 0.5, respectively. Only the fuzzy logical procedures constrain values into this interval, so a variable declared with real can be used for arithmetic operations as well as for logic.

When a variable is declared in HALO, it yields a new environment equivalent to the old environment plus the new definition. If the variable is then disposed in the new environment, the original environment is restored. Thus, if variable x is present in the current environment and is then declared during a subprocedure call, then the subprocedure and the previous environment each have a unique definition of x. This type of binding provides HALO with the scope that LISP lacks. Had the declared variable been present in the original environment, then its definition would have been saved for when the original environment is restored.

Other data types available in HALO are the object and the attribute which are included for modeling and automated learning. The procedure object declares its parameter to be an object in the universe of discourse, and the procedure attribute attributes some property X to the object Y. The primitive operations on objects and attributes include logicals for equivalence (discern and indiscern), a logical for possession (has), an operation to learn new attributes shared by a set of objects (classify), and an operation to display the definition of an object (describe).

CONTROL STRUCTURES

Iteration

HALO is a well-structured language in that it has high-level control structures and facilitates a high degree of abstraction. HALO control structures are built-in procedures which structure their parameters. The control structures, most of which are modeled after Pascal, include the iteratives while, repeat, and for as well as the conditionals if-else, if, and meta-if. The brackets enclosing nontrivial F-expressions eliminate the need for a sequential composition delimiter.

Consider, for example, the HALO while procedure which is expressed as:

\[ \text{while } e_1 \text{ e}_2 \]

where \( e_1 \) is a test expression. As long as the value of \( e_1 \) exceeds the threshold of truth, the expression \( e_2 \) is evaluated. As do all F-expressions, the while procedure returns a value, in this case the final value of the expression \( e_1 \), which must be logically false upon exiting the loop. The repeat structure is also similar to Pascal, and the for structure is as in C.

Conditionals

The general conditional structure in HALO is the if-else procedure which is similar to the if..then.. found in both Pascal and C. A more unusual feature is the meta-if, a conditional preemptive mechanism defined on the state and the store of the program or knowledge base. A meta-if is invoked when there is a change in the state or the store which satisfies its premise. The body of the rule is an F-expression which may invoke another procedure, thus creating "layers" in the knowledge base (Clark and Kandel 1990b). Layered abstraction is discussed in a later section.

Blocks

Procedures and control structures in HALO are applicable only to single expressions, not to lists. The begin procedure allows for bracketing a sequence of expressions to be evaluated and returns the value of the last expression.

\[ \text{begin } e_1 \cdots e_n \]

This is similar to the begin ... end construct of Pascal and allows several expressions to be evaluated in a procedure such as the while. Unlike Pascal, blocks in HALO may be used anywhere an expression is allowed. For example,

\[
> [\text{real count}]
1
> [:= count 5]
5
> [while [> count 0]
1> [begin
2> [write count]
2> [:= count [- count 1]]
2> ]]
5 4 3 2 1 0
>
\]

ABSTRACTION

HALO provides three methods of abstraction: procedures, classes, and layers. Besides using the primitive procedures, the programmer can define new procedures which are either interpreted by HALO or are compiled C or assembly language. New primitives may be defined in C and then added to the language. The following procedural abstract is similar to the while example in the previous section:

255
> [procedure CountDown [X]
1>   [begin
2>     [real temp] [= temp X]
2>     [while [> temp 0]
3>       [begin
4>         [write temp]
4>         [= temp [- temp 1]]
4>       ]
2>     [dispose temp]
2>   ]
1> [CountDown 3]
3 2 1 0
>
HALO procedures may also be defined recursively:

> [procedure Factorial [X]
1>   [if-else [= X 1]
2>     1
3>     [* X [Factorial [- X 1]]]
2>   ]
1>
> [Fact 17]
355687414628352.000000
>
A specialized type of procedure definition, the class
definition, is taken from SIMULA and is especially
useful in structuring large programs (Tennent 1981).
Classes in HALO are procedures whose bodies and
invocations are definitions (Tennent 1977).

HALO also supports layered abstraction, a
bottom-up development technique in which upper
layers preempt the execution of lower layers to alter
their data or execution. The code below demonstrates
the use of a meta-rule to control a lower layer.

> [real i]
1> [meta-if [= i 1] [halt]]
1> [= i 5]
5>
> [while [<= i 0]
1>   [begin
2>     [write i] [= i [- i 1]]
2>   ]
5 4 3 2

The meta-rule preempts the execution of the while loop
as soon as i is set to one (before it can be compared to
zero and displayed).

PARAMETERS

Parameter passing in HALO is by name as in
ALGOL 60, and "lazy evaluation" is employed (Henderson and Morris 1976). In C and Pascal, when an expression
appears in a parameter slot for a procedure, the
expression is evaluated and the procedure is called with
the result. In "lazy evaluation", however, the actual
parameters are not evaluated when the procedure is
invoked, but rather the F-expressions appearing in the
parameter list are substituted literally for the corre-
sponding parameter identifiers throughout the pro-
dure body prior to evaluation. Since nontrivial F-
expressions are already enclosed in brackets, no addi-
tional enclosures are necessary (as in the ALGOL
definition).

In a system that passes by name, procedures can
effectively be called with values, locations, procedure
names, or other symbols. It is important, however, to
remember the "types" of parameters expected. For
example, the parameter X in procedure CountDown
above is not evaluated until its value is assigned to temp.
The CountDown example shows how to effectively pass
arguments by value in a system that only allows passing
by name. Passing by name also allows procedure names
to be passed like other parameters. Consider:

> [procedure func [X] [* X 2]]
1> [procedure one-less [F N]
1>   [~ [F N] 1]]
1> [one-less func 3]
5
>
THE INTERPRETER

Besides evaluating F-expressions, the HALO
interpreter manages all I/O, detects and handles errors,
manages the stack and heap, executes external code,
and interfaces with the operating system. Its principal
task is to obtain expressions to evaluate by prompting
the user for input with the "->" prompt as seen above. If
an expression exceeds a line, the interpreter prompts
again, this time with a number before the prompt to
indicate how many levels of brackets are needed to
close the expression. Since HALO reads from standard
input and writes to standard output, I/O may be re-
directed at the system level. Additionally, HALO is able
to read and execute files from the command line.

When an error occurs, the interpreter backtracks
to the top level, displaying an appropriate error
message at each level. Control is then returned to the user
at the top level. The resulting sequence of error mes-
ages forms a complete execution trace from the level of
the error all the way back to the user command which
initiated the evaluation. The error trace can be manipu-
lated in several ways to allow interactive error correc-
tion and dynamic redefinitions. This mechanism, in
conjunction with various supporting utilities, seems to
substantially accelerate development.

256
HALO PRIMITIVES

The following subsections list the primitives of the HALO language. One of the advantages of HALO, however, is that it allows new primitives to be defined in C or assembly language. Hence, the following procedures may be thought of as constituting the smallest subset of the language.

Logical Procedures

and or not = < > <= >= <> discern indiscern has

Arithmetic Procedures

+ - * / trunc

Input/Output Procedures

describe read write ' (quote)
writeIn tab

Process Control Procedures

halt system load

Binding Procedures

const real object :=
attribute classify dispose ,

Structural Procedures

if if-else meta-if while for repeat begin procedure class

Miscellaneous

(* (comment) help (onscreen help)

HALO is written entirely in ANSI C and is therefore highly portable. Current implementations on personal computers outperform comparable LISP systems in applications which are not highly recursive in nature, especially as the programs increase in size. The HALO interpreter, only a fraction of the size of LISP and PROLOG systems, is well-suited to AI development on small computers and can be used effectively to solve a diversity of classes of problems.

REFERENCES


CONSTRAINT-BASED MODELING OF BEHAVIORS

Jennifer Burg, Charles E. Hughes and J. Michael Moshell

Computer Science Department
University of Central Florida
Orlando, Florida 32816

ABSTRACT

Constraint-based programming is a declarative mode of expression which allows a programmer to describe what an object is rather than what it does. In posing a problem with discrete constraints, we define the problem hierarchically in terms of the relations among its parts. In modeling a real-world object with continuous constraints, we decompose an object into interrelated subsystems. We propose to use both discrete and continuous constraints to model objects with "pluggable" parts, placing these objects together in a scenario to interact in accordance with their constraints.

MODELING WITH CONSTRAINTS

Our work in the past year has been directed towards designing graphical military training systems. This entails modeling battlefield terrain and vehicles, and simulating the actions of tanks and earthmovers. In this context, we have been investigating the usefulness of constraint-based programming in the description of objects and their behaviors.

Modeling a real-world object is a task whose complexity grows in proportion to the level of detail required. Just as a top-down functional design simplifies a programming task, a top-down part-subpart design facilitates the modeling process. A constraint-based description of an object proceeds in this manner. A constraint describes a relation among a subset of the parts of an object, i.e., a subsystem. Two constraints may overlap in that they share parts. Thus, although constraints describe local relations, when taken together they imply global properties (Leifer 1988).

Consider modeling an object as complex as a human being, for example. Because of its complexity, we understand the human body primarily in terms of subsystems: the circulatory system, the pulmonary system, and so on. The circulatory and pulmonary systems are linked, however, in that they both have reference to the bloodstream. This movement from local relations to the global system is a key idea in the application of constraints.

We should note that while most programming is a procedural expression of a problem, modeling with constraints is declarative; instead of saying what an object does, we say what an object is. Left to itself, an object described with constraints doesn’t do anything. What makes it all work is a constraint satisfaction system which can deduce global behaviors from local relations. The constraint satisfaction system finds a consistent state for all the parts of the object. Then, should the object be disturbed from its consistent state, the constraint satisfier will attempt to right it. The constraint satisfier sees to it that an object always "does its own thing."

DISCRETE AND CONTINUOUS CONSTRAINTS

A constraint is a relation defined on a set of objects. Basically, a constraint describes the respective states which a set of objects can be in simultaneously. In the graph coloring problem, for example, neighboring vertices in a graph are constrained to be different colors. Thus if vertices 1 and 2 are adjacent, 1 can be green while 2 is yellow.

The above is an example of a discrete constraint. That is, the states (colors) for the objects (vertices) are chosen from a discrete, finite list of possibilities. Sets of discrete constraints are generally satisfied with backtrack-searching techniques such as those used by Prolog’s inference engine.

Constraints can also be defined in the numeric, continuous domain. The equation $F = 32.0 + 1.8 \times C$ describes a relation between Fahrenheit and Celsius degrees. As a constraint, this equation does not represent a procedure for computing Fahrenheit degrees given Celsius. It instead defines a relation which must always hold true, and it is up to the constraint satisfaction system to satisfy the constraint if either $F$ or $C$ is altered. For this one
constraint, the constraint satisfier could use algebraic manipulation or value propagation techniques. If the set of constraints constitutes a system of simultaneous equations, the constraint satisfier may use numerical relaxation or other equation-solving techniques.

The first step in our research has been an attempt to understand the relationship between discrete and continuous constraints and to formulate problems which require both types. In the remainder of this paper, we will report some of our observations on this score.

**THE CONSISTENT LABELING PROBLEM**

The problem of satisfying constraints in the discrete domain has been formulated as the consistent labeling problem (Haralick and Shapiro 1979). The problem is comprised of a set of units $U$ to which labels must be assigned. The subsets of mutually constrained units are given in the set $T$. The set $R$ tells which combinations of labels are legal for the mutually constrained units.

The blocks puzzle of Figure 1 is an example of a consistent labeling problem, formulated in Figure 2. The task is to fill a rectangular area with a number of blocks. The units to be labeled are the five puzzle slots. The possible labels are the six types of blocks. Each label can be used any number of times in the labeling.

![Figure 1](image)

**BLOCKS PUZZLE AS CONSISTENT LABELING**

$U = \{1, 2, 3, 4, 5\}$; $L = \{A, B, C, D, E, F\}$

$T = \{(1,2), (2,3), (3,4), (4,5)\}$

$R = \{(1, A, 2, A), (1, A, 2, C), (1, A, 2, E), (1, C, 2, D), (1, E, 2, B), (1, E, 2, F), (2, A, 3, A), (2, A, 3, C), (2, A, 3, E), (2, B, 3, D), (2, C, 3, D), (2, D, 3, A), (2, D, 3, C), (2, D, 3, E), (2, E, 3, B), (2, E, 3, F), (2, F, 3, B), (2, F, 3, F), (3, A, 4, A), (3, A, 4, C), (3, A, 4, E), (3, B, 4, D), (3, C, 4, D), (3, D, 4, A), (3, D, 4, C), (3, D, 4, E), (3, E, 4, B), (3, E, 4, F), (3, F, 4, B), (3, F, 4, F), (4, A, 5, A), (4, B, 5, D), (4, C, 5, D), (4, D, 5, A)\}$

**Figure 2**

$T$ tells which slots are side-by-side and thus are mutually constrained. $R$ tells which blocks can fit together in these slots. The tuple $(1, E, 2, F)$ states that a type-$E$ block can fit in slot 1 while a type-$F$ block fits in slot 2. The problem is to assign labels to all units such that none of the constraints are violated.

The blocks puzzle is implemented as a Prolog program in Figure 3. Aside from its "extra" features, Prolog is essentially a constraint satisfaction system. The program has sets of units (variables), labels (variable domains), and tuples defining constraint relations (clauses). The constraint satisfier resides in Prolog's inference engine. The search begins with the query 2- puzzle $(P1, P2, P3, P4, P5)$ and ends in 16 solutions.

![Figure 3](image)

**NUMERIC CONSTRAINTS AND THINGLAB**

The idea of slots into which objects are inserted calls to mind the design of an object programming language, notably Smalltalk (Goldberg and Robson 1989). In Smalltalk, an object is defined by the
instance variables which make up its parts. Relating these parts by means of constraints seems a natural extension of the language. Borning implemented just such an extension in ThingLab, a constraint-oriented simulation laboratory (Borning 1981; Borning et al. 1989; Freeman-Benson et al. 1990). ThingLab is a system for maintaining numeric constraints on objects. Generally, these objects are geometric shapes constructed from points and lines and manipulated through a graphical interface.

Figure 4 is an example of a subclass of ThingLabObject, ThingLab's abstract class for constrained objects. In keeping with Smalltalk, a class of objects is defined in part by its instance variables. To this definition, ThingLab adds numeric constraints placed upon subparts of the object. Note that each constraint predicate must be accompanied by methods which satisfy the constraint locally.

When a ThingLabObject is perturbed, the constraint satisfier is invoked, the constraints on the object are gathered, and the local constraint methods are ordered into a global procedure which will accomplish the desired changes, while maintaining the constraints. Once the object has learned how to change in this way, the method is compiled and put into the object's method dictionary. Subsequent constraint satisfaction is quickly handled by this compiled procedure.

MIXED DISCRETE AND NUMERIC CONSTRAINTS

We have looked at an example of a Prolog program for a discrete constraint problem, and we have discussed ThingLab's object design for numeric constraints. There are in fact systems which handle both types of constraints. Notably, CLP($\mathbb{X}$) is a constraint logic programming language which defines constraints in the domain of real numbers (Jaffar and Lassez 1987). Built upon a Prolog foundation, the CLP($\mathbb{X}$) interpreter uses Prolog searching techniques, augmented with an incremental equation solver.

CLP($\mathbb{X}$) begins with a logic programming basis and adds numeric constraints. What happens if we work from the other direction, beginning with a system designed for numeric constraints (ThingLab) and adding to it discrete ones? Are there any benefits to the object-oriented approach? What does an object approach reveal to us about the nature of discrete and continuous constraints in modeling? We explore these questions in the remainder of this paper.

THINGLAB AND DISCRETE CONSTRAINTS

Consider again the blocks puzzle in Figures 1 and 2. We have translated this into ThingLab style in Figure 5. To represent this problem, we must decompose the object into subparts from the top down. The top level in the part-subpart hierarchy is the Puzzle class. This object is broken down into left-end, neighboring, and right-end blocks. The instance variable le is constrained to be filled with an instance of the LE class. Similarly, nb1, nb2, nb3, nb4, and nb5 must be Neighbors, and re must be an RE. Now the classes for the subparts must be defined.

When we define the LE, Neighbors, and RE classes, we find that all their instance variables can be filled with primitives; we require no further decomposition. Because we are dealing with discrete constraints, the primitives here are symbols.

We have already seen that the instance variables of a ThingLabObject are constrained to be filled with an instance of a prescribed class. Each subclass of ThingLabObject has a prototype. When an instance variable is constrained by class, a copy of the class's prototype is placed in the instance variable.

We can make a simple modification to ThingLab to accommodate discrete constraints. We propose that the modeler build a number of instances of a ThingLabObject subclass, as many as needed to
**MIXED CONSTRAINTS IN THINGLAB**

What if we want to drop below the level of the symbolic block, to numeric primitives which might describe relative color, size, or position? We still want to create instances of the Neighbors class by filling the instance variables `lblock` and `rblock` with actual objects, but the objects in these instance variables are the unique prototypes of the BlockA, BlockB, etc. classes. These prototypes can be further decomposed into subparts filled with numeric primitives. Why would we want to drop down to the numeric primitives, and what happens when we do?

If a type A block is represented simply by the symbol #A, there isn't much we can do with it. Within the problem description, the only meaning associated with the symbol is derived from its relation to other symbols as stated in the constraints. The only understanding which the inference engine has of the symbol lies in its ability to recognize when two symbols match and to logically propagate the assignment of symbols from one constraint to another. The symbol may have additional meaning to the programmer. In his mind, the symbol #A refers to a block of a certain shape. But none of those details are communicated if only a symbol is used.

The symbol is monolithic. It has no subparts, at least none that are relevant to the problem. If the symbol is associated with a graphical presentation, it is an iconic one, with no parts to be accessed. An instance of the BlockA class, on the other hand, could be made up of points, lines, and colors -- parts which can be manipulated by the user in a graphical display.

Numeric primitives are essential to graphically presented simulations, which require objects changing over continuous time and space. Numeric constraints can be used to describe physical phenomena and can be the basis for a visual and aural presentation of a modeled object whose behavior we wish to observe. Continuous numeric values can also be fine-tuned, allowing for more subtle differences between subparts.

Let us summarize how discrete and continuous constraints work together in our ThingLab-style system. When a programmer models an object, he specifies the building blocks which comprise a class and the various ways that these blocks can be put together. For our puzzle, an automatic object-builder could offer a solution based upon the part-subpart constraint specification. The user could then adjust the colors and sizes of the blocks, observing the relative changes to his modeled object as continuous constraints are resatisfied.

Ultimately, we envision a system in which objects such as tanks and earthmovers are built from prefabricated subparts. The top-level object finally is a whole Environment object, into which terrain, terrain features, and moveable objects are placed. The modeled vehicles are then driven by trainees involved in battlefield simulations. We have

---

**Figure 5**

represent the different ways in which the primitive parts of this object can fit together. In our example, the modeller would build as many instances of the class Neighbors as there are tuples in the neighbors relation in the Prolog program of Figure 3. Then the instance variable constrained to be filled with an instance of Neighbors could choose from any of these instances.

The part-subpart relations inherent in the object descriptions are in effect discrete constraints expressed in a hierarchical language. Analogously, Prolog allows us to define a relation in terms of subrelations. The puzzle relation is defined by the leftend, neighbors, and rightend relations. Once again we return to the idea of decomposing a system into subsystems via constraints.

ThingLab's constraint satisfaction system can now be augmented to build objects from the description of the part-subpart hierarchy. We need an algorithm for choosing an instance to fill each constrained slot, selecting from among the existing instances of the class to which the instance variable is constrained. The merges in the object description serve to propagate the implications of one choice to the next. Thus the algorithm deduces all the global structures possible for an object.

<table>
<thead>
<tr>
<th>ThingLabObject subclass: Puzzle</th>
</tr>
</thead>
<tbody>
<tr>
<td>parts: le: LE</td>
</tr>
<tr>
<td>re: RE</td>
</tr>
<tr>
<td>n1, n2, n3, n4, n5: Neighbors</td>
</tr>
<tr>
<td>merges: le: lblock = n1,lblock</td>
</tr>
<tr>
<td>n1,n block = n2,lblock</td>
</tr>
<tr>
<td>n2,r block = n3,lblock</td>
</tr>
<tr>
<td>n3,r block = n4,lblock</td>
</tr>
<tr>
<td>n4,l block = n5,lblock</td>
</tr>
<tr>
<td>n5,r block = re:lblock</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>instances of ThingLabObject subclass: Neighbors</th>
</tr>
</thead>
<tbody>
<tr>
<td>instance1: lblock: #A</td>
</tr>
<tr>
<td>rblock: #A</td>
</tr>
<tr>
<td>instance2: lblock: #A</td>
</tr>
<tr>
<td>rblock: #C</td>
</tr>
<tr>
<td>... and all the other combinations of Neighbors</td>
</tr>
<tr>
<td>instances of ThingLabObject subclass: LE</td>
</tr>
<tr>
<td>instance1: block: #A</td>
</tr>
<tr>
<td>instance2: block: #C</td>
</tr>
<tr>
<td>instance3: block: #E</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>instances of ThingLabObject subclass: RE</th>
</tr>
</thead>
<tbody>
<tr>
<td>instance1: block: #A</td>
</tr>
<tr>
<td>instance2: block: #D</td>
</tr>
</tbody>
</table>
implemented a first prototype of this Environment object with a drivable earthmover as one of its subparts.

FUTURE DIRECTIONS

Our exercises in problem representation and our first attempts at implementation have led us to a number of interesting avenues for further research.

First we note that strict adherence to the object paradigm of Smalltalk can lead to inefficiencies in programming, and it is not yet clear that the advantages in "naturalness" of problem representation outweigh the drawbacks in an object implementation. We intend to continue our comparison of CLP(3) and ThingLab as languages for discrete/continuous constraint problems.

Our next step is to implement the ThingLab discrete/continuous constraint satisfaction algorithm, to consider parallelization of the algorithm, and to consider what benefits, if any, are offered by class-subclass inheritance. So far we have not exploited inheritance. In fact, we must circumvent it. For example, if a method for moving one of the vertices of a Triangle is created by the constraint satisfying and placed in the method dictionary of the Triangle class, this method ordinarily would be inherited by the BisectedTriangle class. Such inheritance is inappropriate in ThingLab since, when a vertex of a BisectedTriangle is moved, a point must also be maintained as the midpoint of the bisected side. The class at a lower level of the class-subclass hierarchy may have more parts and also more constraints than its superclass, and an instance of the subclass therefore cannot necessarily do what an instance of the superclass can do.

The notion of a hierarchy emerges in another form in Barr and Witkin's teleological (goal-oriented) approach to modeling (Barr and Witkin 1989). At the top of the hierarchy, an object is defined by its goals. For example, a tank may have a goal of moving to an embankment. At the next level, the tank is seen in terms of its physical behavior, with movement occurring over time. At the third level, the object is a shape, described as a set of 3-D geometric primitives. At the lowest level, the object becomes a 2-D rendered image. Barr and Witkin propose a modeling scheme unified around a constraint formulation at each level of the hierarchy. Their research is now focused on moving from the top level (goals) to the second level (physical behavior) using a technique called inverse dynamics.

Barr and Witkin's scheme is of interest, and we are considering its application to the modeling of self-motivated objects, that is, objects which are not "driven," but which act according to their own principles in response to stimuli from the environment. This may entail a kind of numerical relaxation which no longer takes place behind the scene, but which is represented graphically as a constrained object seeks behaviors which will resatisfy its constraints. Just as numbers bounce back and forth in numerical relaxation, a perturbed object moves from one behavior to another in an attempt to re-establish its equilibrium.

CONCLUSION

When we program with constraints, we see that expressing a problem and modeling an object are equivalent tasks. In the posing of a problem, constraints can be viewed as logical and numeric relations. In the modeling of an object, constraints can be viewed as parts and subparts arranged into systems and subsystems.

Constraint-based programming offers an alternative, i.e. non-procedural, way to describe an object or problem. Future research in constraint programming should consider the issue of "naturalness" in problem representation, as well as look for heuristics which make practical a constraint approach to modeling and problem-solving.

ACKNOWLEDGEMENTS

This research was supported by grants from the Florida High Technology and Industry Council and the U.S. Army's Project Manager for Training Devices.

REFERENCES


Leher, Wm. 1983. Constraint Programming Languages, Addison-Wesley, Reading, MA.

262
A NEURAL NETWORK APPROACH TO THE DETERMINATION OF AQUIFER PARAMETERS.

Abd.Rashid Abd.Aziz, B.S., Kau-Pui V. Wong, Ph.D., P.E.
Dept. of Mechanical Engineering, Univ. of Miami,
Coral Gables, FL 33124.

ABSTRACT

A new approach to the type-curve matching method has been developed which uses the pattern-matching capability of neural networks to determine aquifer parameters. The network is trained with patterns of drawdown data and their corresponding aquifer parameters, which have been derived from analytical solutions. The trained network is then used to associate field data from a particular pumping test to arrive at aquifer parameters. The results obtained from this new approach are in good agreement with published results using other techniques. The present approach automates the process of fitting field data with analytical solutions, filters errors in measured data, and provides immediate outputs.

INTRODUCTION

The basic analyses of well and aquifer hydraulics by pumping tests all depend on an idealized representation of the aquifer, its boundaries and the nature of the stress of the aquifer. Several assumptions are made whenever analytical methods, such as Theis and Hantush solutions (Theis 1935; Hantush 1956), are applied in the analysis of aquifer-test data. The common assumptions are that the aquifer is horizontal and infinite in areal extent, that the groundwater flow is laminar, horizontal and uniform through the whole aquifer thickness, that water is released instantaneously with a decline in head, and that the well discharge is constant. The related aquifer parameters that could be derived from the above methods are the transmissivity, T [L²/T] and the storage coefficient, S.

The Theis solution to the radial flow of groundwater toward an infinitesimally small-diameter well in an extensive confined aquifer is (Theis 1935)

\[ s = \frac{Q}{4\pi T} W(u) \]  

and

\[ u = \frac{r}{4T} \]  

where \( s \) is the drawdown in water level measured at distance \( r \) from the pumping well at time \( t \) in response to the pumping rate \( Q \) in an aquifer of transmissivity \( T \) and storage coefficient \( S \). \( W(u) \) is the dimensionless response corresponding to the dimensionless time factor \( u \) and can be approximated by the finite series

\[ W(u) = -0.5772 - \ln(u) + \frac{u}{2} - \frac{u^3}{24} + \frac{u^5}{720} - \cdots \]  

For the case of a leaky uniform aquifer of infinite extent completely penetrated by a well of infinitesimal diameter, Hantush and Jacob (1955) gave a solution which is

\[ s = \frac{Q}{4\pi T B} W(u, \frac{r}{B}) \]  

where

\[ W(u, \frac{r}{B}) = \int_{r/B}^{\infty} \frac{e^{-x}}{x} \, dx \]  

is the well function for leaky aquifers and \( B \) is the leakage factor. For \( B \to \infty \), i.e., when leakage \( \to 0 \), Eq. (5) reduces to Eq. (1).

The analyses of aquifer-test data using the analytical solutions are dependent on the comparison of the observed behavior of the groundwater level (drawdown data) with the theoretical behavior in a perfect aquifer, as defined by the Theis equation or its variants. The type curve and the time-drawdown data, when plotted on logarithmic-scale graph paper, should match at all points if the assumptions given by Theis (1935) and Hantush and Jacob (1955) are satisfied for the respective cases.

Effects of barrier boundary, partial penetration, finite well radius and variable discharge for the pumped well cause departure from the theoretical curves. Careful selection of the analytical models is very important in interpreting aquifer-test results and proofs of suitability of the models ultimately rests on field investigations (Reed 1980).
Theis and Hantush solutions for nonleaky and leaky aquifers are selected for analysis in the current work because there are readily available published results where these analytical methods have been applied to actual aquifer-test data and have been proven to produce adequate aquifer parameters. In the new approach, superposition of the time-drawdown data with the analytical solutions are automated by training a neural network to recognize patterns of analytical drawdown data at a specific measurement schedule in a particular aquifer-test design. The parameters predicted by the network are found to be in good agreement with results using other methods.

PRINCIPLE OF NEURAL NETWORK AND ITS APPROACH

Neural networks is a class of artificial intelligence that has been identified as appropriate for pattern association tasks. The network employed in the present work uses a form of supervised learning known as back propagation that learns by having error signals propagate backwards through the network, modifying the network to reduce errors. Training a neural network consists of presenting input-output pairs to the network repeatedly and adjusting the network's internal connections until it produces the correct output for each of the inputs.

The approach is based on the assumption that the theoretical curves will match at all points the field data obtained from aquifer tests. The analysis presupposes a uniform aquifer geometry. Therefore, effects that produce deviation from analytical solutions, such as barrier boundaries, could not be detected by the network trained with the analytical solutions. However, if a significant portion of the measured data falls within the analytical solution, i.e., the effects are felt much later during the drawdown measurement, the network should still be able to predict accurate results.

The effect of well storage, on the other hand, is critical over the first few minutes of a pumping test. Initially, the pump is removing water from the well storage. The drawdown is therefore less than it would be predicted from the aquifer characteristics (Clark 1988). This can seriously affect a pumping-test analysis, because the Theis type curve is at its steepest and is most susceptible to change during the first few minutes. Therefore less weight should be given to the early part of the data. The matching of the analytical solutions with the field data by the network approach is shown to be governed by the fit of the data measured at later time in the measurement schedule.

The effect of leakage to the drawdown curve in a confined aquifer could be seen as a decrease in the rate of drawdown data measured at later time until a steady-state condition is reached. The deviation from the nonleaky type curve depends on the leakage factor. A network trained with the drawdown data from analytical solution calculated with various leakage factors is found to be capable of producing accurate values of the parameters.

One of the advantages of the neural network approach with respect to the current application is its ability to generalize patterns and filter "noise" inherent in a pattern by the network with a known pattern. Measured field data will not match exactly the analytical solutions. This may be due to local variations in an aquifer system within the cone of depression of the pumped well, or due to measurement errors. A trained network will automatically fit the field data with a curve that best represents the data after the deviations have been suppressed.

The automated matching process consists of two steps. The first step is the generation of the training patterns. The patterns are the drawdown derived from assumed values of Q, r, c, T and S. The ranges of T and S are predetermied. For the case of a leaky aquifer, the range of the leakage ratio r/c is also considered. This procedure is simplified by writing PASCAL codes that perform all the calculations and create the training patterns as fact files for input to the network. The second step is the training of the network with the fact files. The patterns are introduced to the network repeatedly until the network is able to recognize each pattern and the corresponding aquifer parameters. The trained network will then be given the field data for evaluation.

DERIVATION OF NEURAL NETWORK TRAINING PATTERNS

The training patterns for the network consists of drawdown data derived from the analytical solutions as input and the transmissivity T and storage coefficient S which are used to calculate the drawdown as output. The generated patterns correspond to a specific aquifer-test design. In an aquifer-test design, the discharge rate Q of the main well and the radial distance r from the observation well to the main well are decided first. A suggested measurement schedule, up to four days, is as follows (Clark 1988):

- 0 - 10 min --- every 30 seconds
- 10 - 60 min --- every 5 min
- 1 - 4 hr --- every 15 min
- 4 - 8 hr --- every 30 min
- 8 - 18 hr --- every hr
- 18 - 48 hr --- every 2 hr

A pattern of draw-down data is derived by first assuming a transmissivity, T, and a storage coefficient, S. With the assumed values of T and S, the radial distance, r, determined from the aquifer-test design and the time, t, from the measurement schedule, the dimensionless time, u, is calculated from Eq.(2). The well function is evaluated next.

264
For the case of a nonleaky aquifer, evaluation of the well function \( W(u) \) is simplified with the finite series representation as in Eq. (3). The drawdown, \( s' \), for the specific time, \( t' \) is then obtained by the substitution of \( q, \ T', \) and \( W(u) \) evaluated at \( u' \) into Eq. (1). For each set of \( T \) and \( S \), a draw-down pattern is generated. A PASCAL code has been written to perform the above calculation on ranges of \( T \) and \( S \) such that these values will encompass the \( T \) and \( S \) of the studied aquifer; on the pretext that the value of \( T' \) and \( S' \) should be within acceptable intervals. The step size of the iterations is an important factor in determining the accuracy of the approach. In using large step sizes, wide ranges of \( T \) and \( S \) could be covered with the sacrifice of accuracy. The drawback in using small step sizes is that a large training pattern is generated, which requires longer computer time in training the network. The best approach here is to use a large step size on wide ranges of \( T \) and \( S \) to obtain the first estimate of the parameters, and then use a smaller step size on smaller range to obtain more accurate results.

Generating draw-down patterns for the case of unsteady flow to a well in a leaky aquifer follows the same methodology as for the nonleaky case. However, the well function \( W(u,r/b) \) is evaluated numerically by logarithmic integration using Simpson's rule, and apart from the ranges of \( T \) and \( S \), the range of \( r/b \) has to be considered also. A similar PASCAL code has been written to generate the training patterns as an input file to the network. Training of the network was performed on an IBM-compatible 8086 microcomputer.

**APPLICATION TO FIELD DATA**

The new approach to the type-curve matching method was tested on three sets of data. The first set of data for a nonleaky aquifer is based on recording by Lohman (1972). These data were analyzed by Kashef (1987) using the Theis type-curve matching method. The second set of data, also for the case of a nonleaky aquifer, is from the 'Oude Krenindijk' pumping test (Kruiseman et al., 1979). This set of data was analyzed by Kruiseman et al. (1979) using the Theis, Chou and Jacob methods, and by Sen (1986) using the slope-matching method. The third set of data is for the leaky aquifer case and the 'Dalem' aquifer test data were selected. These data were analyzed by Kruiseman and de Ridder (1979) using the Hantush (1956) and Walton (1970) methods, by Rushton and Chan (1976) using a discrete numerical model, by Chander et al. (1981) using an iterated extended Kalman filter and by Sen (1986) using the slope-matching method.

The first set of data, consisting of drawdown measurements from three observation wells, was analyzed by Kashef (1987) using the Theis type-curve matching technique with the results of \( T = 0.88 \text{ m}^2/\text{min} \) and \( S = 0.0002 \). Fig. 1 shows the superposition of the Theis type curve with the data from the observation well at the radial distance \( r \) of 121.9 m. The curve is plotted from the analytical draw-down values that correspond to the values of \( T \) and \( S \) observed by Kashef (1987).

![Fig. 1. Theis type curve generated from values of \( T \) and \( S \) obtained from the standard type-curve matching method \( (T = 0.88 \text{ m}^2/\text{min}, \ S = 0.00020) \), superimposed on field data from observation well at \( r = 121.9 \text{ m} \) (Lohman 1972).](image)

The neural network approach was applied to the data from the observation well. Three sets of draw-down patterns with different step size covering different ranges of \( T \) and \( S \) were generated. In the first set of training patterns, the range of \( T \) is from 0.6 \( \text{ m}^2/\text{min} \) to 1.2 \( \text{ m}^2/\text{min} \) with the step size of 0.1 \( \text{ m}^2/\text{min} \), and the range of \( S \) is from 0.0001 to 0.0003 with the step size of 0.0001. Fig. 2 shows two of the training patterns, with constant \( S \) value of 0.0002 and \( T \) at 0.6 \( \text{ m}^2/\text{min} \) and at 1.2 \( \text{ m}^2/\text{min} \). For the same constant \( S \), with variable \( T \), the other training patterns lie in between these patterns. In Fig. 3, two patterns, in which the \( T \) is constant at 0.8 \( \text{ m}^2/\text{min} \) while \( S \) is at 0.0001 and 0.0003, are superimposed on the field data. A total of 27 patterns of analytical draw-down data with the corresponding values of \( T \) and \( S \) were introduced to the network. The network that is trained with the patterns is then tested with the field data from the observation well. The transmissivity, \( T \), and storage coefficient, \( S \), estimated by the network are 0.79 \( \text{ m}^2/\text{min} \) and 0.00022, respectively.

The second set of training patterns consists of analytical draw-down data corresponding to values of \( T \) ranging from 0.7 \( \text{ m}^2/\text{min} \) to 0.9 \( \text{ m}^2/\text{min} \) with the step size of 0.05 and \( S \) ranging from 0.0001 to 0.0003 with the step size of 0.00005. This set contains a total of 25 patterns of analytical draw-down data with the cor-
Fig. 2 Selected patterns of drawdown versus time from Thisi's solution, superimposed on field data from observation well at \( r = 121.9 \) m (Looban 1972). Symbols: \( \bigcirc \): \( T = 0.6 \) m\(^2\)/min, \( S = 0.0002 \); \( \diamond \): \( T = 1.2 \) m\(^2\)/min, \( S = 0.0002 \); \( \times \): field data.

Responding values of \( T \) and \( S \). The network trained with this set of patterns yields transmissivity and storage coefficient values of 0.84 m\(^2\)/min and 0.00019, when tested with the field data. The accuracy of the results could be verified from the plotted type curves that are obtained from substitution of the parameters into the analytical solution, as shown in Fig. 4.

The third set of analytical draw-down patterns was derived from the ranges of \( T \) between 0.74 to 0.94 m\(^2\)/min with the step size of 0.02 and of \( S \) between 0.00012 to 0.00032 with the step size of 0.00002. This set contains 121 patterns of analytical draw-down values and the respective \( T \) and \( S \). These data were introduced to the network. The trained network were then tested with the field data from the observation well. The output results are transmissivity and storage coefficient values of 0.84 m\(^2\)/min and 0.00021, respectively. Fig. 5 shows the Thesi type curve derived from these parameters, superimposed on the field data. A good fit was obtained for the draw-down data measured at a later time. This is a desirable feature since the effect of well storage on the drawdown over the first few minutes of a pumping well could be minimized.

A more challenging analysis would be to test the network with field data that contain more "noise", and compare the results with those obtained from other methods. The 'Oude Korendijk' pumping test data, in particular the data measured at 30 m from the pumped well, was selected for this purpose. Two sets of training patterns were generated. In the first set, the range of \( T \) is assumed from 0.1 to 1.0 m\(^2\)/min with the step size of 0.05 and the range of \( S \) is assumed from 0.0001 to 0.0003 with the step size of 0.00005. The trained network, when tested with the field data, produced transmissivity and storage coefficient of 0.343 m\(^2\)/min and 0.00015 respectively. The second set of training pattern consists of drawdown data generated from ranges of \( T \) between 0.1 and 0.4 m\(^2\)/min with the step size of 0.03 m\(^2\)/min and \( S \) between 0.0001 to 0.0003 with the step size of 0.00001. The aquifer transmissivity and storage coefficient yield by the trained network when tested with the field data are shown in Table 1, along with the parameters obtained using other methods.

Table 1. Aquifer parameters obtained from different methods (using data from 'Oude Korendijk' pumping test).

<table>
<thead>
<tr>
<th>Method</th>
<th>Transmissivity (m(^2)/min)</th>
<th>Storage Coefficient</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thesi</td>
<td>0.230</td>
<td>0.00017</td>
</tr>
<tr>
<td>Chow</td>
<td>0.260</td>
<td>0.00022</td>
</tr>
<tr>
<td>Jacob</td>
<td>0.278</td>
<td>0.00017</td>
</tr>
<tr>
<td>Jacob+</td>
<td>0.333</td>
<td>0.00018</td>
</tr>
<tr>
<td>Slope</td>
<td>0.339</td>
<td>0.00018</td>
</tr>
<tr>
<td>Network</td>
<td>0.326</td>
<td>0.00014</td>
</tr>
</tbody>
</table>

* the observation well is at 90m from the pumped well.

The application of neural network to Hanush type curve for the determination of leaky aquifer parameters is similar to the previous analyses. The training patterns are generated from the ranges of \( T \), \( S \) and \( r/B \). The ranges of \( T \) is taken from 0.5 to 1.5 m\(^2\)/min with step size of 0.2, of \( S \) from 0.001 to 0.003 with step size of 0.001, and of \( r/B \) from 0.1 to 0.5 with step size of 0.1. Testing the trained network with the Dales aquifer test data results in aquifer transmissivity value of 1.368 m\(^2\)/min, storage coefficient of 0.0019 and leakage factor of 770 m. The aquifer parameters obtained from the new approach compared well with those resulted from other methods, as shown in Table 2.
Fig. 4. The frequency curves generated from the outputs of a neural network trained with different parameter step sizes. Symbols: x - field data from observation well at r = 121.9 m (Lohman, 1972).

Table 2. Aquifer parameters obtained from different methods (using data from 'Dalem' pumping test).

<table>
<thead>
<tr>
<th>Method</th>
<th>Transmissivity (m²/min)</th>
<th>Storage Coeff.</th>
<th>Leakage Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hatchen I</td>
<td>1.136</td>
<td>0.0017</td>
<td>600</td>
</tr>
<tr>
<td>Walton</td>
<td>1.201</td>
<td>0.0019</td>
<td>900</td>
</tr>
<tr>
<td>Discrete</td>
<td>1.167</td>
<td>0.0015</td>
<td>850</td>
</tr>
<tr>
<td>Kalman</td>
<td>1.151</td>
<td>0.0017</td>
<td>668</td>
</tr>
<tr>
<td>Slope</td>
<td>1.094</td>
<td>0.0026</td>
<td>505</td>
</tr>
<tr>
<td>Network</td>
<td>1.368</td>
<td>0.0018</td>
<td>770</td>
</tr>
</tbody>
</table>

From the above analysis, it can be concluded that the accuracy of the new approach lies on the step size of the parameters. The best fit of the data is obtained from the network that is trained with the patterns derived from the smallest parameter step size. A very important aspect of the results from the approach is that the output parameters are not in the training patterns. In the analysis, the network adjusted the patterns that it was trained with to the field data, and select the patterns that best represent the data.

CONCLUSION

A new approach to the type curve matching method has been developed which uses the pattern matching capability of neural network to determine aquifer parameters from aquifer test data. The main advantages of the approach are the automated process of fitting field data with the analytical solutions and the network's ability to filter 'noise' in the measured field data. The aquifer parameters obtained from this approach are found to be in good agreement with results obtained using other methods. The accuracy of the fit is also verified by superimposing the type curves generated from the predicted parameters over the field data.

REFERENCES


DISTRIBUTED KNOWLEDGE BASES VIA PARALLEL LOGIC PROGRAMMING

Susan C. Greer, Ralph M. Butler
Susan R. Wallace, Judith L. Solano
College of Computer and Information Sciences
University of North Florida
4567 St. Johns Bluff Road South
Jacksonville, FL 32216

ABSTRACT

In this paper we present an implementation of a logic programming system that provides parallel access to distributed knowledge bases while supporting the semantics of sequential Prolog.

INTRODUCTION

We have implemented a prototype logic programming system which maintains the semantics of sequential Prolog while enhancing it with predicates which support parallel access to distributed knowledge bases. The implementation provides the Prolog programmer with a familiar, yet more powerful, tool for developing distributed knowledge-based systems.

A system may be viewed as knowledge-based when knowledge can be represented non-algorithmically and in such a way that the programs which use it do so by induction and reasoning, rather than computationally. The feature of logic programming which makes it particularly well-suited for development of such systems is its emphasis on the specification of a problem rather than on the algorithm for solving it. Prolog is one incarnation of logic programming which has a well-defined systematic technique for searching for problem solutions. An excellent introduction to logic programming and how it can be used to build knowledge-based systems may be found in (Amble 1987). The typical knowledge-based system uses data which is available locally and does not require remote access. However, there are a number of large applications which do require access to remote data as well as several applications in which the amount of data may be quite small, but it must be distributed. For example, consider the case of a nuclear power plant in which a knowledge-based system is accepting input from several small computers located at strategic points around the plant. These small computers perform sensor readings and communicate their respective local readings to the knowledge-based system which is aware of the global picture and can make recommendations to the local computers about changes which they should make in their local environments.

THE DISTRIBUTED LOGIC PROGRAMMING ENVIRONMENT

Our logic programming environment maintains Prolog semantics in that it provides the programmer with a first-order logic programming language with backtracking. At present, the language supports prefix notation for predicates and Prolog’s list notation. It also has a wide variety of intrinsics which we have found to be sufficient to build any others which were required. The current list of standard predicates includes:

write, consult, listing (and other I/O predicates)
asserta, assertz, retract
sum, sub, etc. – prefix operators for arithmetic operations
lt, gt, etc. – prefix operators for arithmetic comparisons
atomic, integer, var, etc.
unify – prefix version of Prolog’s “=”
fail, true
halt
callgoal (similar to Prolog’s call)
We have found this set of predicates to be sufficient to build all others required. For example, the predicate not can easily be defined in terms of call and fail as:

\[
\text{not}(P) : - \text{call}(P), !, \text{fail}.
\]

\[
\text{not}(P).
\]

Similarly, it is widely known that Prolog's predicates FINDALL and BAGOF can be constructed in terms of CALL, ASSERT, and RETRACT.

The most interesting predicates however, are those that are not available in traditional Prolog systems, i.e. those which support distributed processing. The three basic predicates for distributed computing in our system are:

\[
\text{remotecreate}(N) \quad \text{send}(\text{ToProcId}, \text{Message})
\]

\[
\text{receive}(\text{FromProcId}, \text{Message})
\]

The remotecreate(N) predicate reads a "process group" file that describes the network of processes which the user wishes to execute on a number of processors and returns in N the number of processes that were created. The processes may be referred to by unique integer identifiers in the range 1 to N. The process group file describes which program remote processes are to execute and on what machine each is to execute.

In our distributed Prolog system, the initial process executes a Prolog control loop that provides for interaction with the user. The control loop can either start all remote processes automatically as the system comes up, or the user can create them explicitly from the command line. Each remote process executes a Prolog control loop that communicates with the main process and others as necessary via sends and receives, rather than reads and writes to the user console. The control loops for both the main process and the remote processes are merely Prolog programs that may be easily rewritten by the user.

**THE EXAMPLE DISTRIBUTED KNOWLEDGE-BASED SYSTEM**

In our nuclear power plant example, the main process contains the knowledge base. It receives messages from the remote processes that provide informa-

![Figure 1. Processes Monitoring a Nuclear Power Plant](image)

The body of the control loop for the master process performs the following actions:

1. Receive an update from a remote process
2. Invoke the knowledge-based system to evaluate the updated data
3. If corrective action is required, send required adjustments to the remote processes
4. Return to the control loop recursively

Each remote process executes a control loop that examines sensor readings and communicates with the knowledge-based system about adjustments that need to be made. For purposes of simulation, sensor readings are updated each time through a recursive loop in the program rather than actually being read from an input device. In the demonstration program, each of the remote processes runs on a different computer. We have developed a simple scenario in which one of the computers detects an overload in its portion of the plant and notifies the knowledge-based program, which responds by determining how the correct load should be balanced and notifying one of the other computers to pick up part of the load.
The body of the control loop for each remote process performs the following actions:

- read and update current sensor values
- send current status to the knowledge-based system
- receive message containing corrective action (if any)
- perform corrective action if any was specified
- return to the control loop recursively

**PORTABILITY**

The distributed logic programming system described here is built upon a new implementation of the portable parallel programming package described in (Boyle et al. 1987). Thus, the entire system is portable to both shared memory machines as well as true distributed environments, e.g., a network of workstations. This high degree of portability for the full system insures that programs written for it will also be highly portable. One of the major benefits of such portability is that the distributed data can actually reside on a number of different types of hardware. For example, the knowledge-based system may reside on a large "mainframe" and the remote processes may reside on smaller machines such as scientific workstations.

**FUTURE WORK**

We believe that the distributed logic programming model presented here generalizes to a wide variety of applications. For example, we have developed a project that demonstrates its utility in distributed relational data base systems. We have also started a project to demonstrate how the system can be used to support the more traditional parallel logic programming models, e.g. AND and OR parallelism. In both projects, the additional functionality is built around the three distributed processing predicates REMOTECREATE, SEND, and RECEIVE.

The details of the message passing, process synchronization, etc. however, are hidden within Prolog procedures which are accessible to the user.

**BIBLIOGRAPHY**


NEURAL NETWORK SIMULATION ON A LOCAL AREA NETWORK

Dr. Harold K. Brown, Donald D. Cross, David Lange
Electrical Engineering Department
University of Central Florida
Orlando, Florida 32816

KEYWORDS: Neural networks, local area networks, parallel processing

ABSTRACT

This paper presents a method of simulating neural network topologies and algorithms, using workstations on a local area network (LAN) as a parallel processor. A discussion of the advantages of this approach over traditional simulation methods is included. In addition, a specific neural network topology is discussed which has been successfully implemented on a LAN.

INTRODUCTION

When experimenting with neural network architectures, researchers frequently confront the problem of finding existing hardware on which to run simulations with sufficient memory and speed for real-world problems, since starting with a hardware implementation is time-consuming, error-prone, and generally impractical. Even small problems, when solved with neural networks, can lead to large requirements for processing power and memory utilization (Brown 1990b). Typically, the researcher must choose from expensive mainframe time for high speed and sufficient memory, or slower, small memory microcomputer systems, which are more affordable. However, a local area network can provide a large amount of processing power and memory without requiring unusually expensive or exotic hardware. To meet the speed and memory requirements of a neural net simulation at a comparatively low price, a parallel processor using existing LAN hardware has been implemented successfully.

ADVANTAGES OF USING A LAN FOR NEURAL NET SIMULATION

There are many motivations for the use of a local area network to simulate neural nets, as compared to simulation on single-processor microcomputers. Among these are increased available memory, faster effective processing speeds, and more realistic simulation of an asynchronous parallel system.

Even neural network simulations of modest capability can require on the order of tens of megabytes of data to store all the training data. Such requirements are beyond the memory capacities of most microcomputers. However, on a LAN with such microcomputers as workstations, data can be partitioned uniquely among the workstations, increasing the memory capacity of the system in proportion to the number of workstations. In addition, with a problem distribution topology that has a sufficiently small intercommunication time to processing time ratio, the speed of processing a neural network simulation can also be increased linearly with respect to the number of workstations.

One of the most important properties of a neural network is its massive parallel processing capabilities. Since most neural network simulations of neural networks are performed on machines with only one processor, the parallel processing must be simulated with a sequentially executed set of instructions. However, simulation of a parallel system on sequential hardware can produce misleadingly encouraging results when evaluating a parallel topology or algorithm. For example, a sequential simulation cannot reproduce all the possible variations in synchronization among a group of processors that would occur in an actual parallel system. Since a LAN simulation makes use of physically distinct processing units, such asynchronous effects are inherently incorporated in the simulation, leading to a much more realistic appraisal of the correctness of a given neural network architecture under investigation.

Another simulation consideration is the determination of the relative speed of a neural network architecture in solving a problem. The computation time required to solve a problem using a parallel
algorithm is, in general, not accurately modeled by a sequentially-executed simulation. For example, simulations of two different architectures may yield comparable time efficiencies, while actual parallel implementations of the architectures may prove them to be vastly different in efficiency due to differences in how evenly the algorithms spread the workload among the processors. Again, using a LAN as a parallel processor for neural network simulation yields a straightforward technique for realistic time behavior modeling.

**AN EFFICIENT LAN SIMULATION METHOD**

An efficient neural network simulation using a LAN has been implemented that consists of a central master processor which controls and coordinates a pool of slave processors called cylinders. Each cylinder represents a standard three-layer neural network trained to recognize a subset of the overall solution space for the problem. All of the cylinders taken together embody the complete training set. The training data is split up to make each cylinder responsible for a few of the possible classes of objects to be recognized in the input. For example, in a neural network for recognizing alphanumeric characters in human handwriting, one cylinder might be trained to recognize the letters A, B, and C, while another cylinder would recognize D, E, and F, etc. In this arrangement, the master controller does no actual neural network processing itself. Instead, it merely broadcasts a single image to be classified to all the cylinders at the same time.

When the input image has been transmitted by the master, the cylinders try to classify the image based on their limited partition of the total solution set. Each cylinder thus picks the closest guess from the set of solutions it knows about, and rates the confidence in its guess numerically. The cylinders transmit their guesses and confidence ratings back to the master processor. After the master receives all the responses from the cylinders, it selects the guess with the highest confidence rating as the correct solution.

One problem encountered was normalizing the confidence ratings returned by the cylinders. The training algorithm implemented (Brown 1990a) calculated an error term based on nearest-neighbor solutions. This error term was first normalized to unity, with output magnitudes being assigned in the succeeding layer. The confidence rating was based on this normalized error term rather than the final output.

Distributed processing also introduced a problem with the training algorithm. When run on a single machine, this algorithm works correctly, but when partitioned on parallel processors, problems with its input classification develop. Cylinders with training sets orthogonal to an input would return high confidence ratings. The OETA algorithm creates a basis set from the orthogonal components of a training set. When a new input is presented to the trained network, the nearest neighbor in the basis-set-generated solution space is chosen as the correct output. Originally, inputs completely orthogonal to a trained value would project outside of each component’s endpoints, and due to the boundary conditions given, would generate no error term. A training set containing both the correct input and one nearly orthogonal to the correct input would not exhibit this problem, as the correct input maintains the highest confidence rating. With distributed processing, however, it was possible to have cylinders trained to nearly orthogonal inputs, but not the actual inputs themselves. These nearly orthogonal inputs would yield spuriously high confidence ratings from those cylinders. By changing a cylinder’s solution space boundary conditions, orthogonal inputs were no longer classified as nearest neighbors, and the problem was eliminated.

**COMMUNICATIONS HARDWARE AND PROTOCOL**

The machines used for the LAN neural network simulator were all IBM PS/2 Model 30 systems with Intel 80286 microprocessors, providing processing units with reasonable speed and one megabyte of memory each. The LAN communications hardware consisted of Standard Microsystems Arcnet cards, and the machines ran Novell’s IPX asynchronous communications shell as a software applications programming interface (API) to the Arcnet cards. A custom communications library was developed for the neural network simulator based on the IPX API for sending sequenced packets between machines on the LAN.

Because the individual workstations on the LAN served as self-contained neural networks, there was no need for communication between the cylinder workstations themselves. The only LAN traffic occurred when the master processor distributed the input and when the cylinder processors reported their guesses and confidence ratings back to the master. Thus, the time needed for communication over the network was minimal compared with the time spent running the neural network simulation code on the cylinders in parallel.

**CONCLUSION**

Matching the increasingly widespread use of local area networks in industry and academia, the need for more powerful hardware for neural network simulation is met conveniently by parallel processor implementation on a LAN. Advantages over traditional hardware include a relatively inexpensive hardware base, large memory capacity, high speed, and more realistic assessment of the correctness and performance of neural network designs. Distributed processing of neural network simulations introduces certain complications which do not occur in sequential algorithms, and one must take care to correct these complications.
REFERENCES


A DISCRETE TEMPORAL MODEL OF DYNAMIC PROCESSES IN THE DENDRITIC TREES OF NEURONS

Ryszard Gawronski, Frank Anger, Rita Rodriguez
The University of West Florida, Computer Science Division
11000 University Parkway Pensacola, Florida 32514
(904) 474-2973

ABSTRACT

Research on neuronlike networks has concentrated on a static representation of the short-term response of an individual neural unit. This paper exhibits a dynamic model founded on realistic simplifying assumptions concerning the temporal activity within the dendritic structure of a neuron using an unsophisticated and efficient algorithm. The model is studied via a variety of simple dendritic structures. While establishing that the model is able to provide elementary recognizers for dynamic, spatial, and temporal patterns, the results are discussed. Methods for combining simple recognizers to form more complex ones are also suggested.

1. INTRODUCTION

Most papers on the theory and applications of Neuronlike Networks (NN) assume that the input signals applied to the NN are described by a set of numeric data (vectors or matrices) which are constant during one iteration of signal processing. Moreover, the model of a neuron’s behavior is usually described by an algebraic relation and hence assumed static. The only dynamic processes considered in such networks are those related to the changes, from one iteration to the next, of weights assigned to the input signals due to the iterative learning procedure. However, several authors, among which we find Jan der Heiden 1980, Grossberg 1982, 1988b, 1988c, Arbib 1989, Sejnowski 1989 (including the references cited there), have noted that for some important applications the model of the neuron requires certain dynamic properties, and should therefore be described by a differential equation. In this case, two kinds of dynamic processes can be distinguished:

1. Dynamic processes related to signal processing in the neuron and between neurons. These processes usually have a small time constant.
2. Dynamic processes related to learning procedures (normally iterative) imposing changes of weights in the NN. These processes usually have a large time constant.

Meaningful examples of NNs exist in which input signals are variable. Numerous applications are described by a series of events in which every input event may be characterized by an array of functions depending on time: \( A = \{ A(t) \}_{r} \), for \( r \) in some interval \( (t_0, t_n) \) specifying the temporal extent of the event. For example, recognition of images containing moving objects, voice recognition, and movement perception by interreceptors all exhibit dynamic inputs.

It is well known from neurophysiology (Kuffler et al.1984) that very subtle dynamic electro-chemical processes exist in the dendritic trees (DT) of neurons. Dynamic processes in the main body of the neuron are often modeled (Gawronski 1971, Lewis 1983, Oguzoreti 1983, MacGregor 1987) and included in the description of the NN (Torre et al. 1983, Grossberg 1982, 1988b, 1988c and references cited there, Gawronski and Love 1987). Nonetheless, scanty attention has been paid to the modeling and analysis of dynamic processes within the dendritic trees of neurons, due in part to the very complexity of the structure and processes. In multidimensional cases, the dimension and number of ramifications of the DT is extensive and strongly dependent on the area of the brain in which the neuron is located. Several authors (Rall 1964, Horowitz 1981a, Burke 1987, Chan 1987) suggest that the subtle dynamic processes in the DT are important for the function of the particular neuron. The most convincing example is the detection of the direction of movement by some specific neurons in the visual tract (Barlow and Levick 1965, Marr and Ullman 1981, Veri and Poggio 1989). Other authors have analyzed electrochemical processes in the DT (Rall 1964, Rinzel and Rall 1974, Butz and Cowan 1974, Horowitz 1981b, Pongracz 1985) and presented a mathematical model of a branch of the DT. A single branch may be modeled by a segment of RC cable with losses (i.e. only losses represented by resistance \( R \) and capacity \( C \) of the membrane have to be considered); notwithstanding, a mathematical model of the entire DT is extremely complex and does not lend itself easily to obtaining simple conclusions on the information-processing function of the DT. While some dynamic functions of the DT are relevant, since they are effective in modeling several applications, exact modeling of the DT does not appear to be reasonable.

This paper introduces a simplified discrete model of dynamic processes in the DT, of neurons and a method for analysis of these processes which may lead to some neoteric and exciting learning rules. Specifically, a method of representing the DT as a dynamic operator acting on the input event is shown. Furthermore, the presentation distinguishes the following two kinds of assumptions:

1. Those resulting from specific properties of the DT, for example: delay, damping, elementary inputs treated as pulses.
2. Those simplifying the model and permitting the analysis of concrete structures of the DT, for example: three levels of mutual influence, limited levels of input amplitudes.

With these assumptions, the processes of interest within the DT may be represented as a collection of discrete events whose relationships to one another can be studied using temporal interval analysis. These temporal methods have been developed, among others by (Allen 1983, Ladkin 1987, Anger 1989, 1990, Rodriguez 1990a, 1990b, 1990c). Rodriguez has modified the
branching-time model to a form compatible with the present context. In particular, a "converging-time" model is constructed in place of the usual "linear time," and the relationships between internal events in the DT are consequently mapped onto temporal relations in the converging-time model. Applying the rules of mutual influence as described in the next section, an extraordinarily simple algorithm permits the efficient determination of the resulting output. It will be argued that the discrete model is an appropriate approximation to the actual neural activity and is at the same time rich enough to capture complex dynamic behavior.

Section 2 introduces the description of processes in the DT of a neuron and the temporal model used in creating the algorithm of Section 3. This algorithm captures the dynamic behavior of the dendritic neural structure in response to stimuli. Section 4 presents an analysis of the behavior of the algorithm suggesting applications to visual and aural detection mechanisms. The paper concludes with a synopsis in Section 5.

2. THE DISCRETE DYNAMIC MODEL

A neuron is usually composed of three main parts: a dendritic tree (DT), a main body of the nerve cell, and an axon. Input signals coming from an external source are applied through synaptic connections to the DT and cell body. Output signals are transferred through an axon to other cells. Subsequently, we shall consider only processes within the DT and their influence on the excitation of the neuron.

![Fig. 1 Dendritic tree with excitatory (bar) and inhibitory (circle) inputs.](image)

A dendritic tree may be described by a sketch representing the shape of the tree and positions of all inputs, called synapses (see Fig. 1). Usually there exist many ramifications of thin fibers forming several levels in the DT. Four levels resulting from three ramifications are presented in Fig. 1. Each branch of the fiber from one ramification to the next may be represented by an RC cable (long line) with losses. An equivalent electrical circuit is depicted in Fig. 2. Signal transmission through RC cable, after several simplifying assumptions, may be described by a partial differential equation (MacGregor 1987 p. 36):

$$A^2 \frac{te}{x^2} + Ce + \sum g_i (e - E_i) + I$$

where: $e$ is transmitted potential,
$x$ is the distance from the beginning of the branch,
$A$, $C$, $G$, and $g_i$ are parameters depending on physical properties of the fiber,
$E_i$ is the internal potential of the fiber,
$I$ is the current applied to the branch.

The solution of (1) depends on the applied signal $f(t)$ and boundary conditions. A detailed analysis of solutions of equation (1) may be found in (Horowitz 1981a). Justifications for some simplifying assumptions which will be introduced later may be also found in another paper by Horowitz (1981b). An example of the output signal at the end of a segment of DT, when a short signal is applied to the input, is presented in Fig. 3. As we see the rectangular pulse has been delayed and smoothed, resulting also in a longer tail, consequently justifying various assumptions regarding the discrete model of the processes we present.

![Fig. 3 Response of the segment of DT to a short pulse.](image)

To simplify the notations used in describing the DT it is assumed, without loss of generality, that its structure is that of a symmetric binary tree having $L$ levels. A drawing showing signals and weights associated with each branch is presented in Fig. 4. To model an asymmetric tree input weights corresponding to missing branches should be assumed equal zero.

![Fig. 4 A binary dendritic tree.](image)

There are two kinds of synapses: excitatory (no change of signal sign) and inhibitory (change of the signal sign to a negative value). Several synapses may exist along any branch. The value of the input signal is multiplied by the synaptic weight $w_i$. Each segment of travel from one juncture to the next introduces a
damping and a delay. We may assume that as a result of 
damping the amplitude of every signal passing a branch 
is multiplied by a damping coefficient β.

A short sequence of input signals applied to the DT during 
the time interval [t₁, t₂] will be called an input event. Discrete 
time t = nΔt is introduced for all processes described. Input 
event P is described by a matrix [pᵢ(k)], composed of a set of 
pulses pᵢ(k) of constant duration, where:
1 = 1, 2, ..., L represent the level of the DT where the 
signal is applied (see Fig. 4),
k = 1, 2, ..., K represent the points of input (synapse) 
at a given level K = 2ᵏ⁻¹, and
n = 1, 2, ..., N are the discrete time positions of the pulse 
at the input.

All input signals are positive and have constant amplitude. We 
may assume that the amplitude of the input pulse is equal to a 
convenient integer value, which is then multiplied by a synaptic 
weight.

A signal is propagated along the first branch of the DT 
with a given constant speed υᵣ. Wherever two branches join, 
however, the dendritic branch becomes thicker and the speed 
may jump to a new value until the next juncture. We may 
assume that the thickest (closest to the cell body) branch 
introduces a unit delay, the next one, two units of delay and so 
on. In Fig. 1 the maximum delay within the thinnest branch is 4.

The rules of superposition of signals in a branch depend 
on time relations between pulses. The following rules based on 
simplified description of electro-chemical processes in the DT 
are used (A and B denote two pulses appearing in a branch):

\[ \begin{array}{c|c|c}
A & B & A \\
\hline
\text{t} & \text{t} & \text{t} \\
\end{array} \]

(a) NONE (b) WEAK (c) STRONG

Fig. 5 Temporal aspects of signal interaction.

1. Signals are unrelated if the time between A and B is 
greater than t−influence (tᵢ). Signals continue to 
propagate independently. (See Fig. 5a.)

2. Signals are weakly related if the time between A and B 
is smaller than tᵢ, but greater than 0. (See Fig. 5b.)

As a result, a single signal is obtained with an 
amplitude given as a function of the two signals at 
the time of the second signal.

3. Signals are strongly related when they appear at the same 
time at the same point. (See Fig. 5c.) As a 
result, a single signal is obtained with a larger 
amplitude given as a function of the two signals.

4. If the signal A is inhibiting and appears within tᵢ before the positive signal B, it is always in strong 
relation.

Introduction of these assumptions essentially simplifies the 
description of the dynamic processes in the DT.

3. ALGORITHM DESCRIBING TEMPORAL RELATIONS

Based on the assumptions presented in the previous 
section, and using the method of temporal constraint, an 
operator describing the functions of DT has been constructed. 
This operator transforms the input event P described by a matrix 
[ppᵢ(n)] to an output vector y = (y₁, P described by a matrix 
[ppᵢ(2n)]. This vector may be applied to one or more inputs of the DT of another 
neuron, thus contributing to the formation of the next event 
applied to the second neuron; the discussion, notwithstanding, 
focusses only on the activity of a single neuron. The 
structure of the DT is defined by the following matrix describing 
synaptic weights associated with a specific DT.

\[
W = \begin{bmatrix}
W_{11} & W_{12} & W_{13} & \cdots & 0 \\
W_{14} & W_{15} & 0 & \cdots & 0 \\
W_{16} & 0 & 0 & \cdots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & 0 & \cdots & W_{K1} \\
0 & 0 & 0 & \cdots & W_{K2} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & 0 & \cdots & W_{NN}
\end{bmatrix}
\]

When several inputs (synapses) exist at one branch, an 
appropriate element of this matrix is a vector \( \mathbf{w}_k = [w_{k1}, \\
w_{k2}, \ldots, w_{kJ}] \), where J is the number of synapses existing in 
branch ik.

The matrix \( \mathbf{P}(n) = [ppᵢ(n)] \) describing input signals at 
time n has the same shape as the matrix \( \mathbf{W} \) (including 
the possibility that some elements are vectors \( ppᵢ(n) \) for 
multiple inputs in the same branch), but its elements are binary if it is 
assumed that only pulse inputs are considered. The values of 
this matrix may be introduced successively for consecutive 
values of time n, and does not need to be stored in the computer. 
For every branch ik of the DT, a resulting input signal \( z_{ik}(n) \) 
may be defined:

\[
z_{ik}(n) = \sum_{j=1}^{J} \mathbf{p}_{ik}(n) \cdot \mathbf{w}_{kj}
\]

For representation of dynamic processes under 
the assumptions above on temporal relations, three different values 
of the signal at each branch are defined:

\( x_{curf}(ik) \): the current value of the signal at the node,

\( x_{del}(ik) \): the net value of signals in strong relation with 
the current value,

\( x_{adj}(ik) \): the net value of signals in weak relation with 
the current value.

At any discrete time n the calculation of the output signal 
consists of the successive application of the following formula 
for every level l:1:

\[
x_{curf}(ik) = \beta(\mu x_{del}[-1,k]) + x_{del}[-1,k] + x_{adj}[-1,k] + \mathbf{z}_{ik}(n)
\]

where \( \beta < 1 \) is a coefficient representing strong relations,
\( \mu < 1 \) is a coefficient representing weak relations, and
\( s = 2^{l-1} \) is the second index of the right side branch 
coming to the branch ik. (See Fig. 4),

other notations are as in previous formulas. 
Within each increment of time n, \( x_{del}(ik) \) is replaced by \( x_{curf}(ik) \) 
and \( x_{adj}(ik) \) is replaced by \( x_{del}(ik) \).

For the first layer we have:

\[
x_{curf}(1,k) = \mathbf{z}_{ik}(n)
\]

because only input signals are applied to the first layer. For the 
last layer we obtain for each time n the output:

\[
y_{o} = x_{curf}(l,k)
\]

because in the last layer only one branch L,K exists.

4. APPLICATION OF THE DISCRETE MODEL

As an introductory example of the possibilities of the 
suggested model, a simple DT which detects a moving edge, 
or short pulse, over a line (or plane) of the receptors was 
investigated. According to the idea proposed by Barlow and 
Levick (1964), the structure presented in Fig. 6, was tested by 
means of a computer model based on the foregoing algorithm. 
The computer model allows the introduction of a weight matrix 
after the number of levels L is established. A two-layer (L=3) 
DT was tested first for an edge and a pulse moving from left to 
right and then from right to left.

276
Several structures were tested using Barlow and Levick's general idea of the application of the delayed signal to a synapse having opposite sign. At the beginning we expected that the best results would be obtained for the following structure:

\[
\begin{align*}
+ & \rightarrow \quad + \\
- & \rightarrow \quad -
\end{align*}
\]

A '+' represents an excitatory synapse and a ' - ' represents an inhibitory inhibitory synapse, positioned as in Fig. 6. Verification of some other structures revealed that the best discrimination between the movement to the left and to the right is achieved when the weight distribution presented in Table 1 is applied.

| Table 1 Results at the output of the DT for the structure |
|-----------------------------------------|---------|
| +                                      |         |
| +                                      |         |

Introduce weights for the first layer:

\[
\begin{align*}
w[1][1] & = 1.000000 \\
w[1][2] & = -1.000000 \\
w[1][3] & = 1.000000 \\
w[1][4] & = -1.000000 \\
layer & = 2
\end{align*}
\]

\[
w[2][2] = 1.000000 \quad w[2][4] = 1.000000
\]

Input: An edge shifting right.

Results x(0)[k] for the last layer at successive times:

\[
\begin{align*}
n & = 0 \quad x[3][4] = 0.000000 \\
n & = 1 \quad x[3][4] = 0.000000 \\
n & = 2 \quad x[3][4] = 0.750000 \\
n & = 3 \quad x[3][4] = 1.000000 \\
n & = 4 \quad x[3][4] = 1.562500 \\
n & = 5 \quad x[3][4] = 0.750000
\end{align*}
\]

When an edge moving from right to left was applied to the same structure, the result obtained at the output was identical in magnitude but had opposite sign. The results of the application to the same structure of a short pulse moving to the right are presented in Table 2.

| Table 2 Input: A unit pulse shifting right. |
|-------------------------------------------|---------|
|                                      |         |

Results x(0)[k] for the last layer:

\[
\begin{align*}
n & = 0 \quad x[3][4] = 0.000000 \\
n & = 1 \quad x[3][4] = 0.000000 \\
n & = 2 \quad x[3][4] = 0.750000 \\
n & = 3 \quad x[3][4] = 0.250000 \\
n & = 4 \quad x[3][4] = 0.562500 \\
n & = 5 \quad x[3][4] = 0.187500
\end{align*}
\]

In this case also, a pulse moving in the opposite directions gave identical outputs with the opposite sign. Assuming that the neuron has threshold equal to zero, this structure efficiently detects objects moving to the right. A mirror image of this structure detects an edge or a pulse moving to the left.

Application of the moving signal only to the first layer also gave positive outputs for movement to the right and negative signals for the movement to the left, but output signals were much weaker.

The example clearly shows that the structure depicted as:

\[
+ \rightarrow +
\]

performs differentiation with respect to space of the input appearing at a given time. The next layer sums this effect both in space and in time.

The next experiment was an attempt to find a structure detecting a specific sequence of events. Let us assume, for example, that there exist two sources of sound generating a sequence of short tones denoted as H (high frequency) and L (low frequency). It is necessary to distinguish one sequence, for example HLLH, from other sequences: LLLL, LLHL, LHLH, .... It is easy to see that a full search for an optimal structure for the recognition of this simple sequence requires a very large number of tests for many possible combinations of weights in several layers. Some of these combinations evidently do not permit differentiation. A partial search did not reveal a satisfactory structure for this sequence. Therefore, the solution of this problem was preceded by a detailed investigation of the properties of a simple single-layer DT (L = 2).

To investigate the variety of possible behaviors exhibited by even this simple (single-layer) model, the further assumption that the weights are either +1 or -1 was imposed, and all possible combinations of zero-one input over two time intervals were simulated. The binary nature of the choice for input weights allows only eight different weight assignments in all for the three nodes. Assuming only one synapse at each of the three nodes of the tree, this allows 2³ = 8 different patterns of input in each time period, or 64 patterns of pulses over two time periods.

It is convenient to simplify the array representing the inputs to the dendritic tree to a two dimensional array in which the rows represent the successive time intervals, i = 1, 2, ..., and the columns represent the different input pulses p11, p12, p22. For example,

\[
\begin{array}{cccc}
P_{11} & P_{12} & P_{22} \\
1 & 1 & 1 & 0 \\
2 & 0 & 1 & 0
\end{array}
\]

represents an input to the level-1 branches at time 1 and an input to the level-2 branches at time 2. The effect on the dendritic tree is obtained by multiplying the inputs by the corresponding weights.

The output of the tree is determined by the signal generated in the trunk, which is equal to that at branch 22. Since a single input signal at branch 22 with a positive weight can produce an output signal of 1, the responses which achieve an output with some value greater than 1 were catalogued. By simulating all 3² response patterns, those greater than 1 were found as shown in Table 3.
Table 3. Classification of patterns of input pulses causing an output \( > 1 \).

<table>
<thead>
<tr>
<th>Weights</th>
<th>Input Pulses</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ +</td>
<td>11 12 22 11 12 22 11 12 22 11 12 22</td>
</tr>
</tbody>
</table>

\( T=1 \quad 1 * * \) or \( 1 * * \) or \( 1 * * \) or \( 1 * * \)
\( T=2 \quad * * 1 \)

Any strong or sustained signal

\( + - \)

\( T=1 \quad 1 * 0 * \) Only stimuli starting from left
\( T=2 \quad * * 1 \) with delayed 2nd-level reinforcement

\( + + \)

\( T=1 \quad 1 * * \) or \( 1 * * \) Any increasing signal
\( T=2 \quad 1 1 * \quad 1 1 * \)

All other selections of +1 or -1 weights will give trees with exactly opposite signs from the ones shown. They will produce no output values \( > 1 \) but will produce output values \( < -1 \) in the same patterns as their "opposite" counterparts shown.

Using these results we may find the simplest structure unequivocally detecting a sequence of tones from two sources H and L, for example LH from combinations LL, HL, HH. After the verification of all simple combinations of differentiating structures (that is, structures containing both excitatory and inhibitory inputs in the first layer) the four optimal structures depicted in Fig. 7 were selected.

![Diagram](attachment:image.png)

Fig. 7 Four basic structures of DT enabling the differentiation of four sequences: a) LL, b) LH, c) HL, d) HH.

As we see in Fig. 7 two synapses—one excitatory, one inhibitory—were introduced on the same branch in the second level. Without the introduction of the opposite inputs at the second level, a full differentiation of all combinations was impossible for this simple two-level structure. The distribution of weights is very similar to the structure well known in neurophysiology, called presynaptic inhibition. Selected results of modeling these structures are presented in Table 4, where only the output from the second layer is displayed.

Table 4. Results for structures in Fig. 7

<table>
<thead>
<tr>
<th>a)</th>
<th>Time n = 1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>Outputs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HH</td>
<td>1.5</td>
<td>1.75</td>
<td>0.25</td>
<td>0</td>
</tr>
<tr>
<td>HL</td>
<td>1.5</td>
<td>-0.25</td>
<td>0.25</td>
<td>0</td>
</tr>
<tr>
<td>LH</td>
<td>-0.5</td>
<td>1.25</td>
<td>0.25</td>
<td>0</td>
</tr>
<tr>
<td>LL</td>
<td>-0.5</td>
<td>-0.75</td>
<td>-0.25</td>
<td>0</td>
</tr>
</tbody>
</table>

b) Time n = 1 2 3 4

| Input  | Outputs    |
| HH     | -0.5       | -0.75 | -0.25 | 0  |
| HL     | -0.5       | 1.25  | 0.25  | 0  |
| LH     | 1.5        | -0.25 | -0.25 | 0  |
| LL     | 1.5        | 1.75  | 0.25  | 0  |

c) Time n = 1 2 3 4

| Input  | Outputs    |
| HH     | 0.5        | 0.25  | -0.25 | 0  |
| HL     | 0.5        | -0.75 | 0.25  | 0  |
| LH     | -0.5       | 0.75  | -0.25 | 0  |
| LL     | -0.5       | -0.25 | 0.25  | 0  |

d) Time n = 1 2 3 4

| Input  | Outputs    |
| HH     | -0.5       | -0.25 | 0.25  | 0  |
| HL     | -0.5       | 0.75  | -0.25 | 0  |
| LH     | 0.5        | -0.75 | 0.25  | 0  |
| LL     | 0.5        | 0.25  | -0.25 | 0  |

Comparing with the results presented in Table 3, it is possible to verify that for every structure there is an optimal sequence of signals which is detected by this structure. The discrimination will be more evident in some cases if we additionally assume that the bodies of some neurons have a dynamic capability of integrating over time.

An interesting rule for the optimization of weight distribution may be useful. In selecting a structure for the detection of movement direction, only weight distribution in the first level was optimized. In the second example where simple sequences L and H were discriminated, we used previous results (alternating signs of the weights at the first level) and we optimized only the weight in the second level.

Now we can come back to our previous problem of the structure detecting unequivocally the sequence HLLH. It is simply the sum of structures for HL and LH plus a next level for output. Using the rule above, we may conclude that it is necessary to optimize only the weight distribution in the third layer. Checking four possibilities, it was found that the sum of signals HL and LH should be used in the last layer.

5. CONCLUSION

The preceding sections have shown that a simple discrete model of the dynamic behavior of neurons is possible, avoiding the usual systems of non-linear differential equations. The model is inspired by an interval-based temporal model and an analysis of the electrical activity within the dendritic structure of the neuron. It demonstrates a surprising flexibility under even the most restrictive assumptions. An implementation of the response algorithm clearly indicates that a single two-layer (three-level) dendritic structure is sufficient to create a recognizer for a simple input pattern from dual sources of directional movement across a
visual field. With two layers it is possible to distinguish among the sixteen possible four-bit input patterns and thus among a variety of movements in a visual field. Moreover, a systematic method for building such recognizers from single-layer units is suggested, indicating that the usual highly complex algorithms and slow learning curves may be avoided in a significant class of applications.

5. REFERENCES


Grossberg S.1988b. Neural Networks and Natural Intelligence, The MIT Press, Cambridge MA.


NEURAL BIT SLICE COMPUTING ELEMENT
Joseph Yestrebsky, Paul Basehore, Jerry Reed
Micro Devices
30 Skyline Drive, Suite 200
Lake Mary, FL 32746-6201
Telephone 407/333-4379

ABSTRACT
In this paper we describe a Neural Bit Slice (NBS) building block IC constructed in digital logic. The NBS is intended to be a hardware solution for the construction of synchronous neural networks and for general parallel processing applications. The slice architecture of the NBS allows devices to be interconnected efficiently, allowing many different neural network architectures to be constructed. Networks may be expanded along either rows or columns efficiently by simply adding NBS IC's to the network. Each slice contains eight digital neurons, with 15 “hard” parallel synaptic inputs per neuron. Expansion of (virtual) synapses is possible with modest additional hardware. Synapse expansion is a function of the weight value range and is limited only by internal accumulator overflow. The NBS may be trained under microprocessor control, with 16-bit integer synaptic weights stored in external memory. Processing delay through a single NBS with eight synapses per neuron is 7.2 µsec, providing performance equivalent to that of a 55MIPS processor.

INTRODUCTION
In recent years, the study of neural network models and architectures has seen a resurgence in the engineering community [1]. These models are approximations to the mathematical models of neural activity derived by neurobiologists. The investigative work to date has been limited primarily to computer simulations and rather simple hardware implementations. More sophisticated hardware implementations are limited by the practical problems of packaging large numbers of neurons with the required interconnectivity. This research, however, has established that some interesting practical problems can be solved using these neural network models, particularly in regards to classification, pattern recognition, and image processing.

The advent of inexpensive VLSI/ULSI provides the opportunity to construct low-cost digital neural devices with significant numbers of neurons and synapses. In this paper we describe such a device. The Neural Bit Slice (NBS) is a digital neural network element containing eight neurons. The NBS design allows interconnecting multiple devices to form arbitrarily large networks. The NBS approach is a simulation of a biological neural network, but unlike most software simulations retains much of the parallelism of its organic counterpart. By providing interconnectability of multiple digital VLSI slices operating in parallel, the NBS approach is able to solve for the internal state of many interconnected neurons simultaneously.

The NBS design is independent of network architecture, and device architecture easily accommodates synapse expansion. The NBS contains no training algorithms directly, but may be trained with any appropriate algorithm implemented on an external CPU connected to the NBS. Synapse weights are stored in external memory as 16-bit integers. External memory may be expanded as the network and number of synapses expands. Consistent with a slice philosophy, the NBS is not intended to be a single-chip solution for network construction, but when used in conjunction with some inexpensive “glue” logic may be used to construct arbitrarily large networks. The device is constructed in CMOS technology, contains 22,000 transistors, and is packaged in a 56-pin PLCC.

DEVICE ARCHITECTURE
Figure 1 illustrates the NBS architecture. The NBS is composed of eight individual neurons, plus internal registers and control. Each of eight individual synapse inputs is connected in parallel to each neuron. Additionally, the output of any neuron within an NBS IC can be coupled into any other neuron in the same device through a programmable synaptic interconnect. This effectively provides fifteen “hard” parallel synaptic inputs into each neuron. An external weight memory stores the synapse weights and the neuron bias values.

Registers and control within the NBS allow an NBS-based network to operate autonomously or under external CPU control. Under CPU control the NBS can be “taught” to perform specific functions. In such real-time learning applications, the CPU is responsible for implementing the learning algorithm. For the purposes of training an NBS network, the CPU can access either the dot-product or the thresholded outputs of the neurons through an external CPU bus. New synaptic weights computed by the CPU can then be written into the weight memory (RAM) using the CPU and Register Address busses. Weight storage external to the NBS allows for more efficient network expansion, and reduces the NBS cost and complexity. For applications where the network is trained once and never modified.
This approach is powerful in that it is efficient without increasing processing time in the pipelined data processing systems in common use today.

With the NBS in parallel input mode, each neuron has eight external synapses (8:1 selector), and seven feedback synapses (3:1 selector). When in serial mode, each neuron is connected to only one hard synaptic input. The parallel input mode may be used to provide synaptic expansion, or to "groom" certain inputs (i.e., connect certain inputs only to certain neurons), typically through some external "glue" logic.

Irrespective of the mode selected, Neurons 1 through 8 operate in parallel and synchronously. For each neuron, the output of the steering functions (i.e., the selected input source) is bit-wise multiplied with a 16-bit weight value (AND gate). The results of successive multiplies are summed in an accumulator. The resultant accumulator value is then applied to a non-linear (sigmoid) threshold function whose output is the neuron output. By using a 16-bit weight (fifteen bits plus sign), weight values can range from +32767 to -32768. If the weight values are constrained to the range [+127-128], then using 16-bit weight lengths minimizes the chance of accumulator overflow. Utilizing such an approach thus guarantees synapse expansion to a minimum of 256 per neuron (2048 per IC). (This underscores an important point that in digital neurons where a set of positive and negative values are summed sequentially, the summation results are a function of the summation order. Analog neuron approaches which sum sequentially may also exhibit this phenomenon.)

The following discussion pertains to parallel mode operation. In the context of the NBS, a cycle where all the inputs plus one bias are processed is defined as a frame. Over one frame, each of the eight external inputs is sampled under NBS control, and multiplied bit-wise with the appropriate synaptic weight in the neuron. Thus, to multiply and sum over eight bits requires 8x16 or 128 clock cycles. If feedback is not selected, then the frame is complete. With all seven feedback paths enabled into neuron 1 (and assuming that only thresholded outputs are fed back), the maximum frame time is (8+7)x16 or 240 clock cycles.

Normally, the neuron outputs a new value every frame. Through external control, however, the neuron will update its output only on user-defined input word boundaries. By this technique, synapse expansion is effectively accomplished (up to a guaranteed minimum of 256 for [+127,-
128] weight values), albeit at the expense of processing speed. (Note that this speed penalty does not apply when an NBS network is employed in a 8-bit pipelined application, such as is illustrated in Figure 3). The bias for each neuron is loaded once per frame or input word boundary as appropriate, which adds sixteen clock cycles to the neuron processing time.

The user may select from three possible non-linear threshold functions. All threshold functions are hard-limiting, and are depicted in Figure 4. The double threshold transfer function (#2) is known to minimize the number of neurons and layers necessary to solve some linear inseparability problems (e.g., XOR, parity), and is thus included as an option [3]. Alternatively, the user may choose to pass the accumulator dot-product directly to the neuron output. The dot-product of any neuron may also be fed back as inputs to other neurons in the same device using the feedback mode. This operating mode adds 8x16 clock cycles to the neuron processing time for each frame in which feedback is selected.
When neurons on one layer receive inputs from neurons in two or more NBS devices on the previous layer, one technique for expanding to successive layers is indicated in Figure 5. In this example, the neuron 1 outputs of all NBS devices on a layer may be "wire-and", and so forth for all neurons up to neuron 8, creating an 8-bit bus which can be connected to the parallel inputs of all NBS devices in the next layer. The NBS devices in the previous layer would then be enabled sequentially to communicate to the next layer using some simple control logic. (Disabling an NBS disables only its outputs; the device may continue to process inputs.) This bus oriented approach to layer expansion is efficient in NBS utilization, but increases processing time. In Figure 5, the processing time through layer (X+1) is given as (N+1)x16 clock cycles, where N represents the number of neurons on the previous layer. As well, data cannot be applied to layer X at a rate faster than this. Where processing time is a factor, other more parallel expansion approaches are possible but require more NBS devices.

A large number of different network architectures have been studied, including cascades of two or more differing architectures [1]. Even with this diversity, networks possess only two basic underlying characteristics, feedforward and feedback. The NBS is inherently a feedforward computing element but can also provide feedback efficiently on the same layer, as is necessary when constructing Hopfield min/max networks. Sparingly interconnected networks are also easily constructed with the NBS simply by setting the appropriate synaptic weights to zero. Thus, the NBS is network architecture independent and may be used to construct almost any conceivable network.

**NETWORK EXPANSION ARCHITECTURE**

Network expansion takes three basic forms: (i) synapse expansion, (ii) expansion of neurons on a layer, and (iii) layer expansion. For the purposes of the following discussion, global interconnectivity between neurons in successive layers is assumed. As discussed earlier, synapse expansion may be readily accomplished by external control of input word boundaries. (Alternative parallel approaches for synapse expansion exist which reduce processing time, but require many more NBS devices.) The parallel, globally-connected input structure of the NBS guarantees that each input is connected (by a weight) to each neuron within that device. Expanding neurons on a layer simply requires adding NBS devices to that layer, and connecting together the input busses of all devices on that layer. If synapse expansion were being performed on a layer, then all devices on that layer would be controlled by the same word boundary control signal. Note that expanding a network layer does not increase the processing time through that layer.

**NBS NETWORK PERFORMANCE**

NBS performance can be viewed in both absolute and relative terms. In absolute terms, a 3-layer feedforward network with eight neurons on each layer, fully interconnected, can process an 8-bit input in 21.6us using a 20MHz clock. Each layer exhibits a processing delay of 7.2us. Delay through a layer is independent of the number of neurons on that layer. However, processing delay across a layer does increase as the number of neurons N in the previous layer increases beyond 8. The processing time increases approximately linearly with increasing N.
Comparing the NBS against a conventional CPU must take into account the parallel processing nature of the device. To perform the same logical tasks as a single NBS would require a general-purpose CPU (inherently a sequential function) to perform conservatively 336 instructions per second, or a 55MIPS processing rate. Thus we expect an NBS-based network to outperform many currently available software neural network simulators, and to be more cost-effective than current hardware simulators or hardware-accelerated software simulators.

In terms of neural network benchmarks, the NBS can provide a guaranteed minimum of 256x8 (2048) interconnects per NBS when using a [-127, -128] weight range (with the maximum limited only by accumulator overflow), and a rate of 10 million interconnects per second.

NBS NETWORK EVALUATION

To evaluate an NBS network in hardware requires (1) a physical construct, and (2) a means of training the network. To fill the first requirement, an IBM PC compatible circuit board has been constructed. The board contains two NBS devices and "glue" logic which allows a user to construct simple, arbitrary networks up to five layers deep with up to eight external synaptic inputs per neuron. Control software written for this evaluation board performs layer construction, and allows the user to configure neuron interconnectivity as desired (including intralayer synaptic connections). The board can accept either analog inputs from the external world or (digital) data from an expansion connector on the PC itself. Pre-determined synaptic weights may also be written to the on-board weight memory from the PC.

Since a neural network is of little use if it cannot be trained, a training algorithm must be coupled to the evaluation board. After some study of training algorithms, a Backpropagation (BP) was chosen, since it has mathematical foundation and is guaranteed to converge (at least to a local minimum). Admittedly, BP requires differentiability of the neuron transfer function. The problem is easily overcome, however. If the "signum" network is trained with BP (with sigmoidal transfer functions assumed throughout), a simple weight adjustment at the end of each training cycle can compensate for the true signum nature of the neuron transfer functions.

In essence, BP computes a set of neuron dot-products based on sigmoidal transfer functions, and post-training compensation adjusts the weights for the signum network with the goal of leaving the (BP generated) dot-products unaltered after the compensation. A minor modification to the BP computations is also necessary since we wish that the dot-product of the output neurons be below (for output=0) or above the target (for output=1) instead of converging to the target, suggesting an inequality in the BP algorithm. This modification should increase the degrees of freedom the network has in computing a set of weights that solve a particular problem.

As of this writing, an initial investigation of the modified BP training algorithm has been completed, and a computer program implementing the algorithm is being prepared for integration with the evaluation board. Results on the performance of this algorithm and its comparison to other training approaches will be presented subsequent to further testing.

CONCLUSIONS

We have presented a neural slice computing element whose architecture allows convenient and efficient network expansion. The element implements digital neurons, and provides features necessary to incorporate learning. Device design is independent of network architecture, allowing it to be useful in constructing varying types of architectures. Network processing time increases approximately linearly with network size. The element is inexpensive and possesses a small processing time, so that a network constructed of such elements should be very competitive with other currently available hardware and software approaches.

REFERENCES


Neural Network Learning Using a Constrained Weight Space Search

1Michael Georgiopoulos  1Gregory L. Heileman
1Department of Electrical Engineering
1Department of Computer Engineering
University of Central Florida, Orlando, FL 32816

ABSTRACT

The topic of supervised learning within the conceptual framework of artificial neural network (ANN) models is addressed. An ANN is a parallel distributed processing system that consists of many computationally simple processing elements interconnected through unidirectional weighted connections. Learning in these networks is accomplished through the use of algorithms that adjust the values of the connection weights. The most successful learning algorithm for accomplishing this task, back-propagation, is based on gradient descent error minimization. However, the large amount of training time that the back-propagation algorithm requires is currently one of the factors that limits its use. In this paper, we present an algorithm that requires less time than the back-propagation algorithm to train an ANN.

1 INTRODUCTION

Artificial Neural Network (ANN) models have recently received widespread attention in the scientific community. These computational models, which are loosely based on the anatomy of biological nervous systems, offer potential solutions to problems that have plagued artificial intelligence researchers for many years. The results of a recent study commissioned by the Defense Advanced Research Projects Agency [1] and conducted by the leading experts in the neural network field state that the greatest promise for ANN systems lies in applications that require adaptive learning. The study also found that the ability of ANNs to adapt based on experience might allow them to serve in a broad range of applications more successfully than traditional computational technology. Machine vision, speech recognition and robotics are some of the fields that would most likely benefit from advances in the state of the art of ANN technology. The ability of ANNs to adapt to new environments also has important implications in applications requiring fault tolerance.

Recent studies have demonstrated the pattern recognition capabilities of ANN systems. A major difficulty involved in applying these concepts, however, is in the determination of appropriate learning algorithms for training ANNs to perform their desired tasks. The methodologies used to train ANNs can be categorized into two different learning paradigms: supervised and unsupervised.

Supervised learning requires a set of training patterns of known classification and an external teaching procedure. The teaching procedure is used to adapt network parameters according to the network's response to the training patterns. Normally this adjustment is in proportion to the amount of error present while attempting to classify the current input pattern. The most successful and widely used supervised learning procedure for ANNs is the back-propagation training algorithm [2]. However, the back-propagation algorithm is slow when used to solve difficult problems or train large networks. Adaptations of the back-propagation algorithm have been developed in efforts to reduce the training time involved.

The focus of the research presented in this paper will be on ANN models that utilize supervised learning. Specifi-
cally, a new algorithm named back-propagation with holding is presented that extends the usefulness of back-propagation learning, currently the most popular and widely used ANN learning algorithm. Simulation results are presented that demonstrate the efficacy of this approach.

2 BACK PROPAGATION ALGORITHM

This algorithm accomplishes learning through the minimization of a cost function $J$ that represents the mean square error between the desired and actual network outputs. This cost function can be written as

$$J = \sum_{p} E_{p} = \sum_{p} (T_{p} - O_{p})^{2} \tag{1}$$

where $E_{p}$ is the error associated with the pattern $p$, $O_{p}$ represents the vector of network outputs after presenting pattern $p$ and $T_{p}$ is a vector that represents the desired response of the network when pattern $p$ is presented.

The cost function that is minimized can be thought of in geometric terms as a surface whose topology represents the mean square error of the network outputs for a given set of weights and a given set of input–output pattern pairs. For this reason the cost function is also called the error function. The variable parameters of this function are the network weights. The minimization is achieved by performing a gradient descent through the weight space towards a local minimum of the error function. The magnitude of the weight changes depend directly on a parameter called the learning rate and denoted as $\eta$.

The approach taken in back-propagation is to update the weights after the presentation of each training pattern. This actually implements a close approximation of gradient descent (if the learning rate is chosen sufficiently small).

3 BACK PROPAGATION WITH HOLDING

This algorithm is actually the back-propagation algorithm with pattern holding. In other words, the algorithm implements the standard back-propagation weight update rule; however the input values are held constant at the network inputs until the desired mapping is achieved.

A loose analogy to human learning can be made. Once a person learns a single task very well, he is more likely to remember how to perform that task than if he had to learn this task in conjunction with a number of other difficult tasks.

4 EXPERIMENTAL RESULTS

In the following sections the back-propagation with holding algorithm is tested on a variety of problems and the results are compared with standard back-propagation training.

The Parity Task. The problem that must be solved in the parity task is to determine the parity of an input pattern. If the input pattern contains an odd number of 1's (odd parity), then the output is 1. Otherwise, the parity is even and the network output is 0. In this network, the number of nodes in the first layer is the same as the number of bits in the input pattern. The second layer consists of a single node whose output is used to represent the parity of the input pattern.

The Symmetry Task. Another interesting problem is the symmetry task which requires the network to learn to recognize mirror symmetric input patterns. That is, if the input pattern is symmetric about its center, the output of the network should be 1, otherwise it is 0. Again, the number of nodes, $N$, in the input layer is the same as the number of bits in the input pattern. This network also has a hidden layer that contains two hidden nodes. The final layer consists of a single node whose output is used to classify the symmetry of the input pattern.

The Multiplexer Task. For the multiplexer task, the binary input patterns consist of $N$ bits representing data lines and $\log_{2} N$ bits representing address lines. The network must learn to pass the proper data line to the output according to the address specified by the address bits. The first layer contains $(N + \log_{2} N)$ nodes and the second layer consists of a single node whose output will represent one of the $N$ data lines specified by the address.

The Distributed-to-Local Task. This task is particularly difficult due to the restrictions placed on the network architecture. An $N$-element binary pattern applied at the
input is mapped to a $2^N$-bit pattern at the output. The output bits are all set to 0 except for the one indicated by the input pattern. For example, the 3-bit input pattern 010 (a binary value of two) would map to the output pattern 00100000. Thus, the network must be trained to translate a distributed representation at the input into a local representation at the output. A single hidden node on the second layer receives the activation values of the $N$ nodes on the first layer. Therefore, this hidden node must learn to distinguish between the $2^N$ possible input patterns. The outputs of the second layer are passed to $2^N$ nodes in another hidden layer which in turn feeds into $2^N$ nodes in the output layer.

5 RESULTS–DISCUSSION

Both of the algorithms discussed in the previous sections were simulated on the four benchmark problems presented above. For each of the benchmark problems 25 different (i.e., different initial weight settings) networks were simulated. In our simulations, we took $N=2, 4, 8, 16$ for the parity, symmetry, multiplexer and distributed-to-local tasks respectively. The results of these simulations are displayed in Tables 1–4. We can see from the results that back propagation with holding performs in most cases better than the back propagation algorithm.

It appears that back-propagation with pattern holding outperforms the standard back-propagation learning algorithm if the network is able to adequately retain the information obtained from completely learning a given training pattern set. If this approach is to be successful, the representation formed when learning the previous input-output mapping must not be completely destroyed when learning the input-output mappings of current and future training sets.

The ability to retain the representation obtained from completely learning a given mapping during subsequent training is directly related to the network's information storage capacity. Networks whose nodes are required to make fine distinctions between a wide variety of training patterns will not benefit as much from back-propagation with pattern holding as would networks that are not limited in this manner. This conjecture is supported by the simulation results. Except for the distributed-to-local task, back-propagation with pattern holding clearly outperforms the standard back-propagation algorithm. The network architecture of the distributed-to-local task allows for a single node in the first layer. Therefore, the small number of weights associated with the connections to this node must be trained to distinguish between every pattern class. In such cases, back-propagation with pattern holding would not be expected to perform as well as it does on less restricted networks. This results from the fact that a slight perturbation in these network weights will have a much more profound effect on previously stored information than would a similar perturbation on a network containing a larger number of nodes in this layer. The results shown in Table 4 obtained from the simulation of this task indicate that this is indeed the case.

A heuristic explanation of why pattern holding during the training of ANNs can increase convergence rates is described with reference to the cost function of Equation (1). This overall cost function is computed as a summation of individual cost functions (i.e., the cost functions associated with each training pattern set). It is obvious that the global minimum of the overall cost function is also a global minimum of every cost function associated with a specific training pattern. The back-propagation algorithm focuses on the minimization of the overall cost function. This is a formidable task, for certain problems, due to the fact that individual cost functions may force weight changes towards competing directions. On the contrary, back-propagation with holding focuses on individual cost functions — finding the global minimum of these cost functions is much easier than finding the global minimum of the overall cost function. Why this tends to increase the convergence rate is demonstrated with the help of the Venn diagram shown in Figure 1.

The space $S$ in the diagram of Figure 1 represents the entire weight space (all possible combinations of weight values) for a network being trained to represent three training pattern sets. The space $P_1$ represents the weight values that guarantee the mapping required by training set 1. That is, $P_1$ is the region of convergence for training pattern 1. Notice that the intersection of these regions of convergence yields the set of weights that solves the mapping

286
problem for all training patterns simultaneously,

\[
\text{overall convergence region} = P_1 \cap P_2 \cap \ldots \cap P_i
\]  

The back-propagation with holding algorithm holds pattern \( i \) at the network inputs until the weights are within region \( P_i \). This guarantees the minimization of the individual cost function associated with training set \( i \) and restricts the minimization procedure to the region in weight space denoted by:

\[
P_1 \cup P_2 \cup \ldots \cup P_i.
\]

Such an approach may very well bring the weights close to values that will also represent the global minimum of the individual cost functions associated with other pattern sets of interest, as well as the desired overall global minimum. In contrast, at any point during the execution of the back-propagation algorithm, the weights may assume the values at any point in the space \( S \).

A set of simulation results that supports the conjectures made above is shown in Figures 2 and 3. In Figures 2,3 the number of training patterns learned for the multiplexer task is plotted versus the iteration number. For the back-propagation simulation, shown in Figure 2, nearly 1500 iterations are executed before a significant number of patterns have been learned. For the back-propagation with pattern holding, shown in Figure 3, it is demonstrated that by learning a single training pattern completely, the convergence rate is significantly decreased due to the fact that a large number of additional training patterns are also learned.

6 CONCLUSION

The most successful and widely used ANN supervised learning procedure is the back-propagation algorithm. However, the prohibitive cost of this procedure sets a practical limit on the size of the network that can be trained. This has led to the investigation of methods for improving the algorithm’s convergence rate. One approach, presented in this paper, is the back-propagation with pattern holding algorithm of standard benchmark networks. It was found that the pattern holding resulted in faster convergence rate on a number of benchmark problems.

References


<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Number Solved</th>
<th>( \eta )</th>
<th>Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Forward</td>
<td>Backward</td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>SD</td>
<td>Mean</td>
</tr>
<tr>
<td>BP</td>
<td>24</td>
<td>2.0</td>
<td>15382</td>
</tr>
<tr>
<td></td>
<td>22</td>
<td>3.0</td>
<td>12969</td>
</tr>
<tr>
<td></td>
<td>22</td>
<td>4.0</td>
<td>25071</td>
</tr>
<tr>
<td></td>
<td>22</td>
<td>5.0</td>
<td>25381</td>
</tr>
<tr>
<td></td>
<td>24</td>
<td>7.0</td>
<td>9090</td>
</tr>
<tr>
<td>BPH</td>
<td>25</td>
<td>9.0</td>
<td>538</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>11.0</td>
<td>540</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>13.0</td>
<td>919</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>15.0</td>
<td>537</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>17.0</td>
<td>1581</td>
</tr>
</tbody>
</table>

Table 1: Simulation results of the exclusive-or task for the BP and BPH algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Number Solved</th>
<th>( \eta )</th>
<th>Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Forward</td>
<td>Backward</td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>SD</td>
<td>Mean</td>
</tr>
<tr>
<td>BP</td>
<td>25</td>
<td>1.0</td>
<td>15382</td>
</tr>
<tr>
<td></td>
<td>23</td>
<td>2.0</td>
<td>10285</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>1.0</td>
<td>10682</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>2.0</td>
<td>6314</td>
</tr>
<tr>
<td>BPH</td>
<td>25</td>
<td>3.0</td>
<td>4627</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>5.0</td>
<td>3295</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>10.0</td>
<td>1663</td>
</tr>
<tr>
<td></td>
<td>24</td>
<td>7.0</td>
<td>4432</td>
</tr>
</tbody>
</table>

Table 2: Simulation results of the multiplexer task for the BP and BPH algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Number Solved</th>
<th>( \eta )</th>
<th>Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Forward</td>
<td>Backward</td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>SD</td>
<td>Mean</td>
</tr>
<tr>
<td>BP</td>
<td>6</td>
<td>2.0</td>
<td>29406</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>4.0</td>
<td>27057</td>
</tr>
<tr>
<td></td>
<td>5.0</td>
<td>21058</td>
<td>8748</td>
</tr>
<tr>
<td>BPH</td>
<td>25</td>
<td>4.0</td>
<td>4564</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>5.0</td>
<td>4273</td>
</tr>
<tr>
<td></td>
<td>22</td>
<td>6.0</td>
<td>21058</td>
</tr>
</tbody>
</table>

Table 3: Simulation results of the symmetry task for the BP and BPH algorithms
Table 4: Simulation results of the distributed-to-local task for the BP and BPH algorithms.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Parameter Solved</th>
<th>( \eta )</th>
<th>Iterations Forward</th>
<th>Iterations Backward</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Mean SD</td>
<td>Mean SD</td>
<td></td>
</tr>
<tr>
<td>BP</td>
<td>25</td>
<td>0.5</td>
<td>10801 2312</td>
<td>10478 2112</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>1.0</td>
<td>15337 1276</td>
<td>1230 1214</td>
</tr>
<tr>
<td></td>
<td>24</td>
<td>2.0</td>
<td>3246 856</td>
<td>3047 704</td>
</tr>
<tr>
<td>BPH</td>
<td>25</td>
<td>1.0</td>
<td>8807 2107</td>
<td>8293 2003</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>2.0</td>
<td>6896 1911</td>
<td>6286 1708</td>
</tr>
<tr>
<td></td>
<td>21</td>
<td>3.0</td>
<td>14698 12584</td>
<td>12178 10589</td>
</tr>
</tbody>
</table>

Figure 1. Venn diagram representation of the weight space of an arbitrary neural network.

Figure 3. The number of patterns correctly classified at particular instances during the training of a network to perform the multiplexer task (6 inputs) using back-propagation with pattern holding.

Figure 2. The number of patterns correctly classified at particular instances during the training of a network to perform the multiplexer task (6 inputs) using back-propagation learning.
A DYNAMIC GIGACONNECT NEURAL NETWORK ARCHITECTURE

Harold K. Brown and David F. Lange
Department of Electrical Engineering
University of Central Florida
Orlando, Florida 32816

KEY WORDS
PROTOCOL, TREE STRUCTURE, FEEDFORWARD PROCESSING

ABSTRACT
A neural network system has been developed which allows for over a billion interconnections using a hierarchical architecture. The system is composed of small neural networks that can operate in parallel, with full addressability between them. Processing and communication among the networks are dictated by a sequencing protocol, while connection information is maintained within network data structures. The components of the system are presented, along with an example feedforward processing operation.

INTRODUCTION
To complement the growing sophistication of both neural network algorithms and the types of problems put forth to them, a dynamically configurable gigacommute architecture has been developed (Brown 1990a). This architecture is based on a tree structure, with identically structured units placed in a hierarchical arrangement. Units in the same level of hierarchy communicate with one another via a bus located at their parent's level. Through a sequencing protocol, each unit in the same level of hierarchy takes turns broadcasting the outputs of its children, while sibling units monitor the bus and latch any data required by their children. The simultaneous broadcasting and capturing of data represents the interconnections between the units, and is controlled through data structures located within each unit. The sequencing protocol implements a feedforward cycle of broadcasting, capturing and processing, and is present on each level of the hierarchy.

SYSTEM OVERVIEW
The gigacommute architecture can be described as a hierarchical tree structure of operationally identical units, or Artificial Neural Network Interconnect (ANNI) units. Each of these units communicate directly to a level just above, called the parent level, and a level below, the children level. In this tree organization, a unit and all its 'siblings', are considered children to a common parent. Before any more details concerning the interconnection of these units are discussed, the basic structure and functionality of a unit will be established.

The objective of the ANNI unit cell is to capture input from an external source, present the input to its children for processing, then broadcast the results. The structure required to execute this consists of an input and output list pair, output and pipelined input data space, a bus connecting children units, and access to a parent bus. The structure is utilized during the operation phase of the ANNI unit cell, the 'Feedforward Cycle'. This sequencing protocol channels the data from children of one unit to children of another.

The capturing and broadcasting of data between units occurs over bus transfers, with each unit on the bus occupying a specific address space. A unit captures data by monitoring the parent bus for addresses which fall within its address space. Broadcasting involves placing output data, along with a destination address, on the parent bus. The data may have to pass through several layers of hierarchy, through capturing and rebroadcasting, before the final destination is reached. The broadcasting of input data and output results between several neural networks running in parallel has been demonstrated on a Local Area Network (Brown 1990b). An ANNI unit uses the child bus when it presents captured data to its children for...
processing. In terms of a neural network system, data is captured or directed towards processing via an interconnect to some source. These interconnects are defined in the unit's input and output lists.

The input list contains the addresses of inputs to a unit, while the output list indicates which outputs require broadcasting to other units. The dynamic nature of the architecture is embedded in this list concept. By changing these lists, the actual interconnection is altered. One hardware realization places these lists in a ram data structure, where higher order processing can modify them. The list sizes are reduced by using receiver based logic, where outputs that are required by other units are broadcast globally, and it is up to all units monitoring the bus to decide if the data is to be received. Using transmitter based logic, the output list contains all addresses of units which require the output. Generally, there are many more possible receivers than there are outputs, therefore, receiver based logic is preferred.

The format of the input and output list elements facilitate DMA bus operations using source, destination and count values. For the input list, the source will be the address of the input required, while the destination will point to a place in the pipelined input data space. In the output list, the source contains the address of the output being broadcast, while the destination points to the output in the output data space. In both lists, the count contains the number of addresses to be transferred. To broadcast all the outputs of a typical neural network cylinder, only one entry in the output list would be required. The source and destination pointers would contain the addresses of the cylinder output layer and data, while the count would be set to the number of outputs.

In such a system, care must be taken to prevent sibling units from simultaneously broadcasting on the same parent bus. A protocol has been developed which sequences through all the sibling units, ensuring unique ownership of the bus. All children of a unit are connected in a linked list, with the ANNI unit serving as both the head and tail of the list. The unit sends a broadcasting command to its first child, causing the child to broadcast its data on the child bus. The broadcast command is passed through the list until it is received by the ANNI unit again. The process of broadcast sequencing is of primary importance to the ANNI unit flow of operation, the Feedforward Cycle.

The Feedforward Cycle consists of the Capture state, the Broadcast state and the Process state. All sibling units are initialized to the Capture state. The first of the sibling units is eventually forced into the Broadcast state by a broadcast message from the parent. During the Capture state, a unit monitors both child and parent buses. By matching elements in the output list to addresses on the child bus, a unit fills up its output space, or data to be broadcast. The pipelined input data is received when the parent bus address and current input list element equate. The Capture state continues until both lists are satisfied.

The Broadcast state will be entered only with both the reception of a broadcast message from a back-linked sibling, and the filling up of the output data space. During this state, the unit broadcasts all data in the output space to the destinations found in the output list via the parent bus. When complete, the unit sends a broadcast message to the next unit it is linked to, continuing the broadcast sequencing. After the input and output lists are complete, the Capture state ends and the Process state begins. The first operation of the Process state is the shifting of the pipelined input data, making it available to its children for processing. In all but the lowest level of the hierarchy, this involves broadcasting the data on the child bus. The Feedforward Cycle is then initiated in the unit's children with a broadcast message to the first child. The Process state then exits, and the cycle repeats itself with the Capture state.

These units, although similar in structure, have a definite precedence. To demonstrate the operation of an entire system, the ANNI units will be described as connected in a three level tree structure. The root of the tree is the rack ANNI unit, or the overall neural network system. The rack gathers inputs from the outside world and returns the final results. The level below the rack is composed of units called modules, with the children of a module being the cylinders. The cylinder can be thought of as a basic three layer neural network. While the following describes a simple rack system, the tree can easily be enlarged by running several racks as siblings, tying them together through another level of hierarchy.

The rack system tree structure is diagramed in figure 1, with arrows showing connection paths, and dark lines indicating buses. A typical rack system may contain 16 modules, each with 256 cylinders.

![Figure 1. Rack System Tree Structure](image)

**FEEDFORWARD PROCESSING**

The feedforward operation begins with the top level
of hierarchy, the rack unit, having both received its inputs and broadcast its previous outputs. The cylinders and modules both are at the beginning of their feedforward cycle, with the cylinders having just started their pipelined processing. This processing continues in parallel throughout the feedforward operation, as new inputs are pipelined. The cylinders for each module pass through a broadcast sequence, with their outputs being captured by both sibling cylinders and their parent module. At the module level, if all outputs specified by the module output list are captured from the child cylinders, and an output message has been received, the broadcast state is entered. This output data is broadcast to both the sibling modules, as well as to the rack unit.

Because the rack unit is defined as the top level of hierarchy, its inputs will be available immediately. Once the rack unit receives all its outputs from the modules’ broadcast sequence, it can enter the process state, and shift its inputs down where the children modules can use them. This is actually another broadcast operation, and can be thought of as the last broadcast in the modules’ broadcast sequence.

The module broadcast sequence, along with the broadcast of the rack input, will complete all modules’ input list requirements, sending them into their process state. The feedforward operation is concluded when each module shifts its inputs to the child cylinders. Like with the rack, shifting involves broadcasting the input to the cylinders, filling up each cylinder’s pipelined input for the next processing.

**SIMULATION RESULTS**

A computer simulation to describe the feedforward processing in a rack system is under development. The simulation allows for a variable number of modules and cylinders, along with configurable input and output lists. While it is difficult to precisely simulate asynchronous events, logic operations and throughput analysis can be obtained. The throughput study will involve determining delays for feedforward processing through the cylinders. Delays vary depending on the cylinders’ input sources, as well as the level of hierarchy. Those cylinders which receive inputs from other cylinders within the same module can begin processing before cylinders which require inputs from outside the module. Other factors, such as differences in bus speeds at the rack and module levels, will also be noted.

Intercylinder communication within the same module avoids the overhead of module to module bus transfers. For complex neural network problems, different modules may be trained for specific events, therefore the intercylinder communication would most likely be restricted to the same module. The sequencing protocol as described is optimized for this type of operation, but is flexible enough to be modified to favor module to module communication.

**CONCLUSION**

The giganet architecture presented here provides for a large hierarchical neural network system with dynamic interconnections. By defining the cylinder interconnections in a look-up table data space, dynamic reconnection or modification is possible. A protocol ensures the proper sequencing of operations between elements, including initiating bus speed data transfers between units. The architecture is being simulated on a computer, and the validity of the protocol sequencing logic and complete feedforward processing is being verified. Performance is being collected which will characterize the system for various interconnects.

**ACKNOWLEDGEMENTS**

The authors would like to extend their appreciation to Project Manager for Training Devices (No. 1339–88–G–0002 order 0009) and the Florida High Technology Counsel for funding this effort at the University of Central Florida.

**REFERENCES**


ON THE APPLICATION OF FUNCTIONAL PROGRAMMING TO NATURAL LANGUAGE PROCESSING

S.J. Murrell\textsuperscript{1}  R.T. Plan\textsuperscript{2}

\textsuperscript{1}Department of Computer Science
\textsuperscript{2}Department of Computer Information Systems
University of Miami
Coral Gables
Florida, 33124

ABSTRACT

The aim of this paper is to illustrate that the existing programming languages used in the development of artificial intelligence systems are in one way or another inadequate. The paper proposes that if there exists one language that has all of the advantages of the existing individual languages but none of their disadvantages then it would provide a valuable vehicle for rapid development of A.I. systems. Such a language would allow programs to be reasoned about in a formal way and should be efficient in development time and implementational overheads. The paper then attempts to demonstrate that a language meeting these requirements does exist; it is an advanced functional programming system similar to a number of well-known languages [McCarthy62] [Gordon79] [Turner85], which is extended to incorporate parallelism and other valuable features. As an illustration, a small intelligent database information retrieval system is implemented.

Keywords: Functional Programming, Natural Language Understanding, Parallel Processing.

Introduction

The aim of this paper is to consider the applicability of functional programming systems to the development of intelligent database retrieval systems.

The first section of the paper demonstrates that the existing standard languages used in the development of A.I. systems are each in some way inadequate, and that these languages are highly unsuited to the development of applications in the field of user interfaces that involve natural language processing. We further illustrate that the traditional languages do not facilitate formal reasoning of their behavior.

The second section of the paper discusses a language that is functional in nature and that possesses none of the disadvantages associated with existing languages (discussed in section one), and as such it will provide a valuable tool for rapid development of A.I. systems.

The advantages of this language are illustrated in section three which shows the language applied to a small database information retrieval system with a natural language interface. We attempt through this example to discuss the advantages of size, formality, style, ease of writing, and maintainability.

Having shown how the use of an advanced functional programming language can be used to significant advantage over conventional forms of programming, the final section of the paper discusses the use of parallel processing within this language to provide a more efficient programming environment.

Languages

An examination of the programming languages normally used for artificial intelligence and expert systems applications, shows them to be divided into several categories

- Procedural (or algorithmic): the 'standard' languages such as Pascal, Ada, or even Fortran.
- Logic based: Prolog.
- Shells: such as Guru, VP-Expert etc.
- Object oriented: such as Smalltalk and C++.

We shall consider each of these categories in terms of the following criteria: program size, code complexity, syntax, semantics, formality and maintainability.

Programs written in procedural languages are typically quite large. Standard languages have a lot of inelegant and unwieldy syntax; every aspect of a program's control structure must be explicitly stated and each language construct typically involves several keywords and other syntactic elements by which it is recognized. This generates larger volumes of code than may be really necessary to state the programmer's intentions.

The complex and intractable semantics of
procedural languages makes some of their aspects occasionally difficult to understand and always very difficult to reason about and manipulate in a mathematically formal and rigorous manner [Hoare69] [Dijkstra76].

The ability to reason about programs in a formal manner is especially important in two respects: the need for the programmer to be able to understand the language he is using, in order to write programs correctly, and the need for that code to be open to mathematical manipulation to make possible proof and transformation. Procedural languages do not facilitate either of these needs, due to their side effects and the emphasis on control structures. Thus, proving a procedural program correct is a very difficult task.

Prolog was designed to overcome some of these difficulties. Prolog has such a limited domain of applicability (making deductions from logic equations) that it can have a much simpler syntax and semantic structure; any problem that can be reduced to the repeated application of rules of logical inference, can be implemented in Prolog with extreme ease. Any problem that requires anything more makes demands of Prolog that it was not designed to meet. Adding extraneous features to a language in order to unnaturally extend its domain, is bound to increase its complexity.

Functional Languages

The language we advocate is a "pure functional language, closely related to Lisp but with a more friendly syntax, which is a strict superset of the L-K variant of Church's Lambda Calculus[Church41], and therefore has already been proved to have all the power of a Turing machine.

Functional languages are not extensively used in artificial intelligence applications because they are commonly considered to be too restrictive, too difficult to use effectively, and in fact, too academic for use in real world applications.

These, however, are unfortunate misconceptions. Admittedly programming in a functional language does require the programmer to think about problems in different ways, and at a higher level of abstraction, but after an initial period of adjustment these factors cease to be problems and become advantages.

The essence of functional programming is its strong basis in pure mathematics. Programs are executable specifications of the desired behavior (in other words a program in its own formal specification) Programs are sets of pure mathematical equations; they have the same semantics, and in many cases the same syntax as pure mathematics. Thus the thousands of years mankind has devoted to the development of the theory of pure mathematics and all the resultant proof and reasoning methods are at the programmers' disposal.

This seemingly restrictive style is both the major advantage and the major disadvantage to functional programming.

As an example of such a program, we present the insertion sort:

In the following, \( t \) is a list constructing operator, \( s \) is a list destroying operator, \( a,b,e \) are variables and \( [] \) represents the empty list.

First we define sort in terms of simpler functions; to sort a list, we simply sort the tail of the list, then insert its head into the result. This gives a naturally recursive definition:

\[
\begin{align*}
\text{sort} \ (ht) &= \text{insert} \ h \ \text{sort} \ t \\
\text{sort} \ [] &= []
\end{align*}
\]

To insert an element in its correct place in an ordered list, we compare it with the head of the list. If it is less than it should be placed before the head, if it is more then it should be inserted in its correct place in the tail:

\[
\begin{align*}
\text{insert} \ e \ (ht) &= \begin{cases} \\
\text{c:ht} \ &| \ e\text{-}h \ | \\
\text{h(insert e t)} \ &| \text{[]}
\end{cases} \\
\text{insert} \ e \ [] &= [e]
\end{align*}
\]

Although at first sight, this four-line program seems difficult to read, once the meanings of the unfamiliar operators are known, it becomes very easy to understand, and it obviously correct. The non-algorithmic nature of the language is demonstrated by the fact that although the program was written as an insertion sort, it runs as a bubble sort.

A significant advantage of functional languages is that they effectively reduce the amount of work that the programmer has to do, by:

\( \text{o} \) Having a natural syntax that allows the programmer to write what he means directly, without being concerned with cumbersome notation,

\( \text{o} \) Obviating the error-prone design of control-flow patterns,

\( \text{o} \) Making the use of infinite or unbounded data structures and functions possible.

The power of functional programming is made apparent by a study of the implementation of Prolog's logical reasoning process. An analysis of Prolog's methods reveals that the system may be decomposed into a five register abstract machine [McCabe83] [Murrell87], which is ideal for direct implementation in a functional language such as we propose. This implementation may be rendered in a single page of code, and originally took less than two hours to design.
Parallel Processing

The purity of functional languages makes the inclusion of parallel processing features simple and efficient. These features also become easy to use.

Traditional parallel programming techniques require the use of such features as Semaphores, Synchronization Points, Shared Variables, Communication Channels, etc. In a pure language, all of these features are forbidden -- How can a language with no variables have shared variables? How can inter-process communication take place if no process may have any effect on any other?

At first sight, these restrictions would seem to prevent anything useful from being done. Parallel processes are 'obviously' no use if they are forbidden from communicating with one-another. However, as with the other special features of functional languages, a simple adjustment of the programmer's expectations is the solution.

Parallelism is an implementation of non-determinism. When there are two (or more) possible paths to a correct solution of a problem, of which some may succeed and some may fail (e.g. when Prolog selects a clause for resolution), an easily understandable method is not to determine in advance which is the correct path, and not to take one path and back-track if it fails, but to take all paths simultaneously and accept the first to reach its goal. This application of parallel processing requires no interaction between processes, is very easy to use, and is compatible with the functional style.

References

Church, A.E. 1941
"The Calculi of Lambda-Conversion"
Annals of Mathematical Studies 6,
Princeton University Press

Dijkstra, E.W. 1976
"A Discipline of Programming"
Prentice Hall
Englewood Cliffs

Gordon, M.J., Milner, R., & Wadsworth, C.P.
"Edinburgh LCF ."
Lecture Notes in Computer Science
Vol 78, Berlin: Springer Verlag.1979

Hoare, C.A.R. 1969
"An Axiomatic Basis for Computer Programming"
Comm. ACM. 12-10, 576-580.

Hoare, C.A.R. 1985
"Laws of Programming"
Technical Monograph PRG-45
Computing Laboratory, Oxford.
VALIDATION TOOLS FOR EXPERT SYSTEMS

Uma G. Gupta and
Dr. John Bliege
Dept. of Industrial Engineering
University of Central Florida
Orlando, FL 32816

ABSTRACT

Several expert systems have been developed for business and industrial applications. However, the ratio of number of systems developed to number of systems deployed is low. One of the major hurdles to deploying expert systems is a lack of formal validation and verification techniques. Expert systems are "intelligent" systems. Since we, as of yet, do not know exactly what we mean by intelligence, how to capture it and how to measure it, evaluating expert systems is a challenging task. This paper is a discussion on existing validation tools and their strengths and weaknesses.

INTRODUCTION

Expert systems have been developed for a number of applications ranging from computer configurations and medical diagnosis to design and process control. Expert systems are particularly useful for critical and real-time applications where timeliness and expertise in problem-solving can make the difference between chaos and competency. In spite of the many benefits of this young and growing technology, the ratio of number of systems deployed are low. One of the major hurdles in deploying expert systems in an operational environment is the inability to evaluate and measure the robustness and performance capability of these systems. Until now, testing expert systems has been mostly ad hoc, informal and arbitrary (Green 87). However, this approach is both totally inadequate and inappropriate for critical, real-time systems. The challenges and complexity of testing real-time embedded systems, as compared to stand-alone systems, increases multi-fold. Rigorous and formal validation and verification tools are essential in order to make expert systems operational and win users confidence. Lack of such tools may seriously hamper the future growth and applications of expert systems.

TERMINOLOGY

The terms evaluation, validation and verification are often used interchangeably in the context of software testing. These activities have different goals and implications in software engineering. Evaluation is an assessment of the overall quality of a piece of software. Evaluation comprises the analysis of a number of factors such as performance validation, code verification, user friendliness, system efficiency and program documentation.
It is a process of evaluating all factors that influence the efficiency and usability of a piece of software. Validation and verification are, therefore, subsets of evaluation. One of the difficulties in evaluating expert systems is the lack of a yardstick against which expert system performance can be compared. This is because we do not, as of yet, know how to identify, measure and grade expertise. Expertise, in many domains, is still largely undefined, immeasurable and non-standardized. Validation addresses the question "Is this the right system?". Is it capable of solving the problem(s) that it was designed to solve? Does it satisfy user requirements? Validation is black-box approach since it focuses only on input-output and ignores the process by which the input was converted into output. Verification, on the other hand, addresses the question "Is the system built right?". Is the code in each module traceable to its specifications and have the specifications been transformed into error-free code. Verification studies the code for sequence, structure and specifications and hence is often referred to as a "paper activity". Verification detects errors that validation cannot and vice versa and hence the two have to be used synergistically.

Evaluation, validation and verification of expert systems pose unique challenges to the software development and test team. While some of the techniques used in conventional software can be borrowed, they cannot be blindly applied to expert systems.

Testing expert systems can be broadly divided into three areas:
1. verifying the knowledge base for completeness and consistency
2. certifying the reasoning paradigms of the inference engine.
3. validating the output of the system and evaluating its performance.

A considerable amount of research has been done in knowledge base verification (Nguyen et al 1987, Suwa 1982, Stachowitz et al 1987). Knowledge base verification involves establishing that the knowledge in the knowledge base is complete and consistent. A secondary issue related to knowledge base verification is ensuring that the knowledge is maintainable (update, add, delete and modify knowledge) and can be refined without undesirable side effects. The inference engine is a piece of conventional software that processes the knowledge in the knowledge base. It can be verified using conventional software techniques. Several powerful expert system shells are currently available and hence developers of expert system applications are rarely confronted with inference engine verification. The area that has been least researched and is unfortunately prone to errors is performance validation. In simple terms, performance validation aims to establish that an expert system is in fact an expert.

Several important issues confront the tester of an expert system:
1. What criteria should be used to evaluate a system? Given that there are no set criteria for measuring human expertise, how does one establish that a system possesses expertise?
2. How do we determine if the system is giving the right answer for the right reasons? In several domains, there is no right or wrong answer but a range of answers that vary from poor to excellent. How do we evaluate those shades of grey especially when we have
never graded the performance of the expert?
3. Is the system giving the right answer for the right reasons?
The knowledge in the knowledge base must be correct and the processing of that knowledge by the inference engine must be correct. However, it is quite possible that the system gives the right answer even though one of these factors (knowledge or inference) is incorrect (sophisticated explanation facilities partially alleviate this problem). The problem is further exaggerated when confidence factors and fuzzy logic are used.
In spite of these challenges and complexities, little research has been done on testing and evaluation of expert systems. Often, the tools and approaches used to validate expert systems are ad hoc, arbitrary, informal and in many cases unscientific. The tools that are currently used to validate expert systems are:
  - Field Test
  - Face Validation
  - Turing tests
  - Sensitivity Analysis
  - Statistical Techniques
  - Test Cases
Some of these techniques are still in the experimental stage while others are widely used. The suitability, strengths and weaknesses of each of these techniques is briefly discussed below.
Field Test: The system is placed in the operational environment and its performance is observed. The burden of testing, to a large extent, rests on users because developers fix problems that users detect. Quite apparently, this technique is unsuitable for critical or real-time systems.
Face Validation: The system is evaluated by a group comprised of developers, testers, users and even domain experts. The system is simply evaluated at its face value. This is an excellent testing technique for a final prototype but is quite unsuited for critical or real-time systems. It is recommended as an informal testing technique.
Turing tests: The performance of the system and that of the human expert for a set of problems are presented to a group of evaluators without disclosing the source that solved the problem. The group evaluates the performance and this evaluation is a means of “grading” the level of expertise of the system. In a good system, the evaluator will not be able to distinguish between the system and its human counterpart. This approach is an excellent way to eliminate computer bias.
Sensitivity Analysis: The input(s) to the system are varied systematically and its effect on the output is analyzed. The effect is the same as writing a large number of test cases with minor variations in the input. Sensitivity analysis, although a powerful and efficient technique, is one of the least utilized validation tools.
Statistical Techniques: Several statistical techniques can be used when comparing the performance of the system with that of the expert. Techniques such as regression analysis, analysis of variance, chi-square tests, Hotelling T squared tests and simultaneous confidence intervals have been proposed as possible validation techniques. However, the value of many of these techniques as validation tools is unknown.
Test Case Generation: Generating test cases and exercising them on a system is one of the most popular validation techniques. Usually, the expert writes a set of test cases that are run through
the system. The performance of the system on the test cases is then evaluated to detect errors. It is a useful and powerful test technique, especially when used in conjunction with other techniques.

In the rest of this paper we discuss the merits and loopholes of using test cases as a validation technique. Test cases are usually designed by the domain expert from whom domain knowledge is acquired. The system is evaluated based on its performance on the given set of test cases. There are several strengths to this approach. It allows the expert to test the finer points of a system by designing test cases that exploit the intricacies of the problem domain. Test cases can be designed to capture problems that users will encounter and this helps to make a system more realistic and user-friendly. A library of test cases can be maintained so that the system can be tested every time a change is made to it.

Unfortunately, this technique has several major drawbacks. To begin with, the expert who is integral to the development of the knowledge base is also the key architect in designing test cases. In other words, the developer is the tester and this undermines the credibility of testing. Second, one of the elements of expertise is the bias that an expert brings to problem-solving. When the expert develops test cases, these biases are further reinforced.

Hence for a system to perform at acceptable levels, it should have the same biases that the expert does! Third, testing is a highly creative activity and the domain expert may not be endowed with the traits of a good tester. There is evidence that experts tend to be biased, prejudicial and parochial while designing test cases and are often not even aware of these traits. Assuming that a good domain expert is also an effective tester is questionable. Fourth, a large number of test cases must be generated to perform even a preliminary test on a medium-sized system. Test case generation is both a time-consuming and expensive activity. Lastly, given the size, complexity and criticality of expert systems, the ability of the expert to test all aspects of system performance by generating test cases is highly questionable. These are some of the reasons why test case generation is grossly inadequate as a stand-alone validation technique.

We recommend a structured approach to testing expert systems. Some guidelines are as follows:

1. Establish measurable evaluation criteria before validation begins. This is more easily said than done. Very little research has been done on software metrics for expert systems. However, some guidelines should be set to determine if the performance of a system is acceptable to the users. Without evaluation criteria, testing gets tied to factors that negatively impact the testing process. Some such factors are project deadlines, reduction in manpower and testing dollars.

2. It is imperative both for the credibility and quality of testing that a domain expert who was not part of the development process be responsible for designing test cases. Therefore, for any system a minimum of two experts are required: one expert will help to build the knowledge base and the other expert will help to test the knowledge base.
3. Gather information on the type and number of each type of case that the expert is usually called upon to solve. Categorize the cases as simple, average and complex and generate test cases that mimic the "real-world". In other words, executing a large set of complex cases on a system that will rarely encounter a complex case is not an efficient or effective way to measure system performance.

4. List a set of validation techniques that will be used to test the system. There is no single technique that can detect all errors in a system. A group of techniques should be used such that the weaknesses of one system is balanced by the strengths of another.

Conclusion:
Evaluating the performance of an expert system is a highly challenging task and testers are faced with several complex issues. The performance of an expert system is compared with that of its human counterpart and is either classified as acceptable or unacceptable. The problem in using this approach is we do not as of yet know how to evaluate the performance of a human expert.

Hence, expert system testing is a subjective process. This does not mean that it has to be ad hoc or informal. Formal, rigorous and structured techniques should be used to establish system reliability. This paper discussed some validation techniques that are currently used with specific emphasis on test case generation. Some recommendations were made to enhance the testing process.

Bibliography:

Chandrasekaran, B. 1983. On Evaluating AI systems for Medical Diagnosis. AI Magazine (Summer). 34-37


A BIOLOGICALLY-BASED NEURAL NETWORK MODEL

Glen Cornell
Department of Computer Engineering
University of Central Florida
Orlando, Florida 32826

ABSTRACT

Despite current trends in artificial intelligence, an obvious methodology to acquire the functional knowledge of intelligent behavior lies through the study of true intelligence. However, of the overwhelming number of models of intelligent behavior, only a small fraction are biologically based. And of these biologically based models, only a fraction are acceptable from a neurophysiology standpoint. An "acceptable" model is one which would discriminate all but a few of the multitude of processes that are known to occur on the neuronal level and label the rest as "noise." On a microscopic level, this model would appear crude because of its lack of functional detail. Yet, it would be safe to assume (Hillis 1988) that, from the simple rules of local interactions of neurons in an organized network similar in structure to the columnar network found in the cerebral cortex (Hubel and Wiesel 1968), the emergent behaviors of the system would seem life-like. Therefore, the following paper presents the two major components of an accurate model of neural automata. The first element, the rules of cellular operation, are based on the evidence leading to independent pre- and postsynaptic neuronal plasticity rules (Carew et al., 1984). The second component of the model presented in this paper is the set of rules governing the structure of the network, which is based on the columnar organization of neuron groups perpendicular to the cortical surface (Mounicaplace 1957).

The scope of this paper encompasses a portion of the theory of neuronal group selection proposed by Edelman (1984). An attempt is made in the theory to explain the functional nature of the topographic receptive maps in terms of Darwin's theory of group selection.

1 THE POSTSYNAPTIC RULE

Intercellular communication, the transmission and reception of neurotransmitters and ionic concentrations, occurs at the synapse. The postsynaptic, or receiving, neuron accepts input through pore-like holes, called ion channels, which regulate the inflow of ions such as calcium and sodium. Short-term learning is believed to be the variation of the channels' efficacy, or ability to allow ions to pass, at the postsynaptic site. The process of modifying the efficacy of the postsynaptic neuron is as follows:

1) The presence of neurotransmitters from the presynaptic cell binds with the postsynaptic neuron, causing channels to change their state (from open to closed, or closed to open) and to generate a modifying substance.

2) Channels can only be modified in the closed state. Therefore, as the concentration of the modifying substance increases, the probability for an unmodified closed channel to undergo a change in its efficacy increases.

3) Ions may flow through open channels, which cause the local postsynaptic potential to deviate from the cell's resting potential. Modified channels open after a refractory period; and more ions consequently flow into the postsynaptic cell.

Derivation of the following postsynaptic rule includes the modeling of three major groups. The first portion explains the nature of the local postsynaptic potential. The second portion of the derivation describes the new cellular voltage. The final component to the postsynaptic rule is the computation of the postsynaptic efficacy.

1.1 Local Postsynaptic Potentials

Starting with the Hodgkin and Huxley model of the neuron (1952), the following schematic is assumed to accurately represent the electrical properties of the postsynaptic area of a true neuron:

![Fig. 1: Schematic representation of the postsynaptic cell.](image-url)

In the Hodgkin and Huxley model, two types of channels (Potassium and Sodium) act as conductances ($g_K$, $g_{Na}$) in parallel to the cell membrane capacitance ($C_m$) and a small leakage current ($i_L$). Evidence (Kandel, 1981) has...
shown, however, that modifiable ion channels, other than Sodium and Potassium, exist at the postsynaptic site. Because the exact number of variations of channels is not yet clear, a generalized derivation of the above equation, assuming that there is only one ionic driving force, simplifies the theory of the model presented in this paper. On the basis of this assumption, we may redraw the circuit by eliminating the small leakage current and all but one ionic voltage source:

![Diagram of modified Hodgkin and Huxley circuit](image)

Fig. 2: The modified Hodgkin and Huxley circuit.

Solving for the postsynaptic current, \( I(t)_{ij} \), at the postsynaptic cell \( i \) receiving input from the presynaptic cell \( j \):

\[
(1) \quad I(t)_{ij} = C M \frac{dV}{dt} + \sum_{i=1}^{N} (E_i - V) g_i
\]

Because the population of channels at one synapse contribute a minute voltage compared to the cell’s potential, the change in voltage of the cell membrane with respect to time is negligible. Assuming, additionally, that all ionic conductances and voltages are equivalent, the resultant equation for the current at time, \( t \), is as follows:

\[
(2) \quad I(t)_{ij} = A (E - V) g
\]

Where: \( A \) = the number of active channels (N).

By Ohm’s Law, the postsynaptic potential can be computed using the impedance of the cell membrane in the following equation:

\[
(3) \quad V(t)_{ij} = Z \cdot I(t)
\]

By substitution:

\[
(4) \quad V(t)_{ij} = Z (E - V) A g
\]

Notice that the term, \( Z (E - V) \), remains constant, while \( A \), the number of active (open) channels and \( g \), the ionic conductance, varies. At another point in time, \( t_2 \), the number of modified channels will be added to the previous number of active channels. Thus, the equation for the postsynaptic potential at time \( t_2 \) is:

\[
(5) \quad V(t_2)_{ij} = Z (E - V) (A^* g^*)
\]

The rate at which the postsynaptic voltage changes with time can be found according to equation (6), which includes two new terms, \( S_j \), the activity or rate of firing of the presynaptic cell, and \( \xi_j \), the presynaptic efficacy or the ability to allow transmitter to be released from the presynaptic cell.

In order to simplify the equation, the constant terms are accumulated into the constant, \( K \), where the subscript indicates the type of the postsynaptic cell:

\[
(6) \quad \frac{dV}{dt} = K_j (A^* g^*) \sum j S_j \xi_j
\]

Where \( A^* \) is the additional number of active modified channels and \( g^* \) is the corresponding modified conductance of the channels. In order to validate the previous equation, the variability in the conductance must be limited to two values, modified and unmodified.

1.2 Net Cellular Voltage

The computation of the net voltage, the summation of positive excitatory and receptor potentials and negative inhibitory potentials found by equation (6), seems plausible. This learning scheme is similar to that proposed by Grossberg (1968) in which a passive decay term, representing the subtraction of inhibitory inputs, was introduced into the “classic” Hebbian Learning Postulate (Hebb, 1949).

However, if nothing but inhibitory inputs were active at one cell, this assumption suggests that the cell’s potential could eventually drive to infinity. This, obviously, is not true because the neuron’s voltage is known to vary anywhere from -120 to +40 millivolts. A simple solution to this problem is to provide the following shunting inhibition term:

\[
(7) \quad \sigma_t = \frac{1}{1 + \exp(K_i \sum_j \eta_{ij} \xi_j S_j)}
\]

This term simulates inhibition through an exponential function (a common sigmoid function) which was chosen rather arbitrarily. A step or ramp function could have just as easily been implemented; however, the goal of this paper is to remain as close to the true nature of cellular function (which does not operate in quantized steps or uniform slopes) despite the seemingly insignificant variation.

As the shunting inhibition prevents the inhibitory inputs from driving the neuron’s potential to infinite inhibition, another process, synaptic saturation, \( v \), prevents the excitatory and receptor inputs from driving the neuron’s potential to an opposite and infinite excitation. The physiological description of synaptic saturation is the decrease in transmembrane ionic driving force as the voltage increases. This is simulated by the following equation:

\[
(8) \quad \sigma_v = \frac{1}{1 + \exp(V_j)}
\]

Thus, the net voltage can be computed according to the following differential equation:

\[
(9) \quad \frac{dV}{dt} = \sigma_t \sigma_v \left[ K_E \sum_{j} \eta_{ij} \xi_j \sum S_j + K_i \sum_{j} \eta_{ij} \xi_j S_j \right]
\]

301
1.3 Postsynaptic Efficacy

Because postsynaptic modification can be voltage dependent (Malinow and Miller, 1986), Calcium dependent (Kandel, 1981), or a myriad of other possibilities which are reviewed by Finkel and Edelman (1987), an adequate model of modifying substance concentration is more difficult to derive. Assuming, therefore, that the concentration of a general "modifying substance" behaves according to a simple flow problem, a constant increase with respect to the voltage level of the presynaptic neuron and a constant decrease with respect to time, the difference equation is as follows:

\[ M_{ij}(t + 1) = (1 - \sigma_{M}) M_{ij}(t) + K_M \sigma_j S_j \]

Where: 
- \( M_{ij} \) = The concentration of modifying substance
- \( K_M \) = Substance growth rate
- \( S_j \) = Activity of the presynaptic neuron
- \( \sigma_j \) = The presynaptic efficacy
- \( \sigma_M \) = Substance decay constant

Knowing that the channels can be in one of two states, open (active) or closed (inactive), and that a channel can be modified only when it is inactive, the following state diagram accurately models the state that the population of channels can be in (Finkel and Edelman, 1987):

- \( A \rightarrow a \rightarrow I \)
- \( b \)
- \( k_{b2} \)
- \( k_b \)
- \( k_f \)
- \( M \)
- \( A^* \rightarrow a^* \rightarrow I^* \)

A = Active or open channels
I = Inactive or closed channels
A* = Active modified channels
I* = Inactive modified channels
a = inactive \( \rightarrow \) active transition rate constant
b = active \( \rightarrow \) inactive transition rate constant
a* = inactive mod. \( \rightarrow \) active mod. trans. rate
b* = active mod. \( \rightarrow \) inactive mod. trans. rate
Kf = Forward modification rate
Kb, Kb2 = Backward modification rates

Fig 3: Channel State Diagram

Noting that channels may be modified in the inactive unmodified state, the following transition equations emerge:

\[ \frac{dA}{dt} = A^* K_{b2} + I a \cdot A b \]

\[ \frac{dI}{dt} = A b + I* K_b - I a - I M K_f \]

\[ \frac{dA^*}{dt} = I^* a^* \cdot A^* b^* - A^* K_{b2} \]

\[ \frac{dI^*}{dt} = A^* b^* + I M K_f \cdot I^* a^* - I^* K_b \]

The postsynaptic efficacy, by substitution, is found as the following:

\[ \eta_{ij} = A g + A^* g^* \]

The cell's rate of firing is determined according to the common "sigmoid" function:

\[ S_i = \frac{1}{1 + \exp(- V_j)} \]

Up to this point, only the effects of short-term learning can be accounted for. The dynamics of the postsynaptic model presented here allow for chemically-induced changes which last only as long as the persistence of the modifying substance. This consequence, however, does not indicate a flaw in the model. On the contrary, in the actual postsynaptic mechanism, changes occurring on the order of 100 ms have been recorded to persist for minutes, hours, and, at most, even days (Finkel and Edelman, 1987). If the postsynaptic mechanism is not responsible for the modifications persisting for years, then the need for another mechanism is the obvious result.

2 THE PRESYNAPTIC RULE

Long-term modifications in a cell's ability to fire are divided into two general categories, facilitation, the increase in transmitter release due to an increase in activity, and depression, the decrease in transmitter release due to a decrease in activity. The different types of presynaptic modifications are described in Table 1 and Figure 4.

Facilitation is known to occur under two circumstances. The first is post-tetanic potentiation, which is an increase in the homosynaptic potential lasting for minutes to hours following a train of high frequency (tetanic) pulses applied to the presynaptic neuron (Kandel, 1981). Homosynaptic facilitation, the second type of facilitation, is what is known most commonly as sensitization, the increase in attentiveness to an otherwise neutral stimulus following the activity in another (Heterosynaptic) pathway (Kandel and Tauc, 1964).

Similar to facilitation, two forms of depression are known, low-frequency depression and presynaptic inhibition. The first form, low-frequency depression, discovered by Lloyd (1949) is described as the decrease in the synaptic potential following a period of low activity. The resulting effect is known as habituation, or the increasing ability to ignore a repeatedly applied stimulus. The final form of depression, heterosynaptic inhibition, occurs when a synaptic potential is decreased by the activity at another synapse in the path (Dudel and Kuffler, 1961).

Because the amount contributed, if any, by the aforementioned forms of presynaptic plasticity to the process of learning is yet uncertain, this paper will generalize the modifications into two groups, facilitation and depression.

<table>
<thead>
<tr>
<th>Table 1: Forms of presynaptic plasticity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Facilitation</td>
</tr>
<tr>
<td>Depression</td>
</tr>
</tbody>
</table>

302
Each group has a homosynaptic and heterosynaptic component corresponding to the general description above.

2.1 Facilitation

The facilitation term presents here accounts for an increase due to both homosynaptic and heterosynaptic cellular activity and a decrease due to a passive decay of calcium ions, which have been found to be essential to the release of transmitter. Notice that facilitation is less likely to vary as the distance of the heterosynaptic transmitter release increases:

\[
\frac{dF_i}{dt} = \mathcal{E} S_i - \lambda F_i + \sum_{j \in H} \frac{1}{d} \xi_j S_j
\]

Where: 
- \( F_i \) = the degree of facilitation 
- \( d \) = the distance from heterosynaptic cell \( j \) 
- \( \mathcal{E} \) = the rate at which calcium flows into the presynaptic cell. 
- \( \lambda \) = facilitation decay rate

2.2 Depression

The depression term follows a somewhat similar pattern as the facilitation term. Notice, however, that depression increases as the amount of transmitter released (Ei Si) increases:

\[
\frac{dD_i}{dt} = \kappa \xi_i S_i - \beta D_i + \sum_{j \in H} \frac{1}{d} \xi_j S_j
\]

Where: 
- \( D_i \) = the degree of depression 
- \( \kappa \) = depression growth constant 
- \( \beta \) = depression decay rate

2.3 Presynaptic Efficacy

In the study of the giant squid axon, Smith (1984) discovered that postsynaptic potentials increased with the third power of the presynaptic calcium current. In other words, a small change in calcium concentration results in a large change in the presynaptic efficacy. The following equation for the presynaptic efficacy reflects the results of Smith's findings:

\[
\xi_i(t) = \xi_i(0) (1 + F_i t^3) (1 - D_i)
\]

Where: 
- \( \xi_i(0) \) = A baseline average in the presynaptic efficacy 
- \( 0 \leq F_i < \infty \) 
- \( 0 \leq D_i \leq 1 \)

Fig 4: Synaptic modifications. Synapses ik and il are homosynaptic, whereas synapses ik and ij are heterosynaptic.

As is shown from equations (17) and (18), the problem of long-term retention remains. Perhaps the simplest way to solve this problem is to include a baseline presynaptic efficacy which the efficacy at any particular point in time may vary from (Finkel and Edelman, 1987). The baseline efficacy is merely a time-based average.

CONCLUSION

The foundation of the paradigm presented in this paper stems from the electrochemical mechanisms found in the nervous system. Yet, the intent of this paper is not to model the particular characteristics of these mechanisms, but rather to generalize the input-output relationships into a collection of rules which can be implemented into an artificial neural network. The functional accuracy of the model supports the belief that a network based on the preceding paradigm will result in enhanced recognition and generalization capabilities over other cellular automata such as the back-propagation and the Hopfield algorithms.

However, generalizations of the electrochemical mechanisms result in obvious limitations. For example, this model is limited to only one modifying substance controlling plasticity; whereas in the neuron, genetic mechanisms allow for changes in the chemistry, if the need arises. Another limitation of this model is the discrete computation of values. Koch, Poggio, and Torre (1983) found that synapses spread throughout a dendritic tree behave in a nonlinear fashion. Because the neuron can make temporal generalizations, the importance of propagation delays of signals in a dendritic tree to information processing is believed to be significant. This belief, however, is yet to be formalized.
REFERENCES


COMPARISON OF AN ANALOG IMPLEMENTATION TO A NUMERICAL SIMULATION OF THE OUSTAR PATTERN LEARNING NEURAL NETWORK.

Erich Nold, Leslie Canney, and Michaeal Georgiopoulos

Martin Marietta Orlando Aerospace Corporation
Image and Signal Processing Engineering and the Technical Computing Center
P.O. Box 5837, Mail Points 1040 and 170
Orlando, Florida 32853

The University of Central Florida
Department of Electrical Engineering and Communications Sciences
Orlando, Florida 32816

ABSTRACT

Both a "nongated", and "presynaptic gated" Oustar (Grossberg 1982), were implemented in fully parallel, real-time, analog (or physical) electronic circuits via an analog computer. Analog results are directly compared to a digital (or numerical) simulation of the identical equations attempted by the analog circuit. Neural signal "activity" is represented by an analog "voltage amplitude" in the model studied. The analog Oustar system exhibited instability of long term memory weights in varying degrees of severity, as a function of the chosen system parameters, and the temporal separation of input "pulses" to the network. A simple model of the analog multiplier voltage offsets, when imposed on a "clean" digital simulation, is shown to model much of this unstable performance. The analog circuit instabilities reported are independent of the specific Oustar circuit design, as they hold for all analog electronic circuits which perform a multiplication of two physically continuous signals. It is hypothesized that this result applies to all analog media, such as optical, electro-optical, or acoustical.

INTRODUCTION AND TAXONOMY

Neural networks are presently being studied by engineers, physicists, biologists, and applied mathematicians. For the purpose of clarity, one may define two broad categories of neural network research.

I. Artificial Neural Networks (ANN).
II. Biological Neural Networks (BNN).

The category of ANN may further be broken into three subcategories.

a. Numerical (ANN) = Discrete, or "digital" network simulation.
b. Hybrid (ANH) = Analog/digital hybrid combination.
c. Physical (ANN) = Purely analog signal processing, whether electrical, optical, acoustic or in combination, such as acousto-optic, or electro-optic.

Note that ANN possess all the common algebraic properties, such as unique mappings for addition, and multiplication. Electronic ANN processes are classically documented (Sheingold 1976) to perform multiplication in a "nonunique" fashion. That is, all multiplications of two continuous analog variables produce a small "epsilon" offset in the product term. This epsilon is a random variable, dependent on both the input variables, and device characteristics.

ANNP is now broken down into two final sub-subcategories.

ANNP-Externally Controlled (ANNPIC)
ANNP-Real-Time (ANNPRT)

"Real-time", is meant as defined by Carpenter (1989) to be those (and only those) system dynamics explicitly defined by the set of O.D.E.'s which define the neural network. "Externally Controlled" refers to any external "corrections", or calibrations imposed upon the system during execution, which are independent of the system O.D.E.'s. Hopfield and Tank (1986) and Kosko (1989) study ANNPRT systems. Mead (1989) reports many ANNPIC system implementations. All three of these research articles utilize a straightforward technique of representing a neural "activity" signal level (defined as a(t) in (1) and (2) below) as a "voltage amplitude".

What if the physical (or "analog") substrate, when attempting real-time operation (ANNPRT), executes something other than the O.D.E.'s intended? Sheingold (1976) points out classical limitations of analog electronic multiplication, which cause ANNPRT circuits of the "activity onto voltage amplitude" type studied below, to exhibit a nonunique (or randomly offset) multiplicative mapping. These inexplicable physical "offset" (or random error) manifestations cause the unstable long-term memory (LTM) characteristics measured, and reported below.

CIRCUIT IMPLEMENTATION DESCRIPTION

The specific single oustar system studied attempts to implement the following equations, with n=5.

\[ \dot{x}_1(t) = -\alpha x_1(t) + I_1(t), \]  
\[ \dot{x}_2(t) = -\alpha x_2(t) + \beta x_1(t-\tau)x_1(t) + I_2(t). \]

\[ y_1j(t) = z_1j(t) \left( \sum_{k=2}^{n} z_1k(t) \right)^{-1}, \]  
\[ \dot{z}_1j(t) = -u z_1j(t) + \beta x_1(t-\tau)x_1(t), \]  
where j=2, 3, . . . , n, and z_1j(t)>0, j≠1, n \sum_{k=0}^{n} x_k(t)>0 (Grossberg, 1967).

Copyright (C) 1990 Martin Marietta Corporation.
All Rights Reserved.
Proceedings of the 3rd Florida Artificial Intelligence Research Symposium, 1990, by the Florida AI Research Society
c. 1990 FLAIRS 0-9620-1732-9/90/0350-0395

305
Equations (1) and (2) are referred to as activity equations or the short term memory (STM) equations. Activity level, or signal level, $x(t)$, at various points in the network is represented as the time varying voltage amplitudes in the electronic circuit. The $x(t)$ signals are simply node voltage outputs. Equation (3) represents a normalization, or competition process, Equation (4) represents the training or learning process invoked. Set (4) are also referred to as long-term memory (LTM) equations because they are associated with the longer term process of adapting the connection or synaptic weights, $a(t)$ and $y(t)$, of the network. Figure 1 describes the graph of (1) through (4). This system is referred to as "nongated" below.

Parameters $\alpha$ and $\beta$ were fixed at 100, and 1000 respectively. The learning decay factor, $u$, was nominally chosen to be $u_0 = 10$. These values were intuitively chosen from the digital simulation. The analog circuit was scaled based on these parameter settings. The $u$ factor was then varied from 0.1 through 50 for this study.

As discussed later, small $u$'s caused the $z(t)$'s to rail off (or saturate), and large $u$'s caused the $z(t)$'s to "bottom out". Bottom out occurs when the $z(t)$'s decay down to such a level that they are lost in the noise floor, and thus become "equal", or washed out. This bottoming out, previously misinterpreted by the author as caused by offsets at the inputs in Nold and Canney (1989), can be noted in Figure 4, of this previous paper.

A second Outstar system, utilizing "presynaptic gating" (Carpenter 1989), was implemented to correct for bottom out. The gated circuit changes (4) so become:

$$\dot{z}_{ij}(t) = [-u z_{ij}(t) + b x_{ij}(t)] x_i(t - \tau) \tag{5}$$

The "pre" or "post" synaptic terminology refers to which term, the outstar source signal $x_1(t)$, or the outstar sink signals $x_j(t)$, are "gating", or multiplying the bracketed term in (5). A postsynaptic learning rule would simply swap the position of the $x_1$ and $x_j$ terms in (5). System (1), (2), (3), and (5) are referred to as "gated" below.

---

Figure 2. Outstar with $n$ nodes. This schematic is the analog circuit which is implemented on the hybrid computer, with $n = 5$. 

306
Equations (6), (7), and (8) more accurately model the analog implementation of (2), (4), and (5) respectively. Digital results of this simple model explain in large part the instabilities measured in the analog system. Sheingold (1976) points out that these "a" offsets are indeterminate, random variable functions of the multiplicative input factors, and device characteristics. Small positive, or negative constants have been chosen for these "a"s due to the extremely (much slower than the LTM time constant) variation of these terms. For example, one offset was noted to change in magnitude by approximately 10% during a 100 second circuit run.

Similar "ε" offset modeling was digitally simulated at the "divider" output in (3) (see figure 1). Study of this extended model is ongoing. Analytical study of the above reported model is also ongoing. The ε-model approximation simply requires a small positive or negative real number to be added to each multiplier, and divider output. This is the primary result of this analog circuit experiment, and its implications are made clear in the following measured circuit performance description.

DISCUSSION OF PURE ANALOG, PURE DIGITAL, AND DIGITAL "ε" MODEL OF ANALOG RESULTS.

When analog scaling is matched to parameter choice, the Oustar performs perfectly during training (see figure 4, 4=9). When training inputs are removed, and consecutive recall pulses are applied to the source node (x(t)=0), the Oustar LTM traces gradually retrain to the offset "suite" which is then present in the circuit. "Suite" refers to the magnitude and polarity of offsets at the multiplier outputs. Offsets at summer inputs, and at integrator inputs have been ignored as these did not markedly effect circuit performance.

Attention was limited to modeling the analog multiplier offsets only, as they are large, measurable, and resultant of, or caused by, the Oustar "feedback process of interest" in this study. Additionally, these offsets appear in portions of the circuit which are unaffected by the addition of thresholds, or sigmoidal nonlinearities as discussed in Grossberg (1982). Both these noise suppressing constructs were tested as written in (9):

$$
\dot{x}_j(t) = -\alpha x_j(t) + \beta f(x_1(t-\tau) - \Gamma) y_j(t) + I_j(t),
$$

where G is a threshold, or deadband, and f is the sigmoidal squashing function (Carpenter 1989). Neither of these constructs effect the multiplier offset problem, because the multipliers are "hidden" within the nodes themselves. The feedback accumulation of offsets is independent of the f and G maps. This can easily be seen in the figure 1 schematic, noting that the sigmoidal f, and G threshold occur during the (t-τ) delay branch of the graph, and so do not suppress the multiplier offsets.

In figures 3-5, "T" denotes a training state (I(t) through I5(t) active), "R" denotes a recall state (I1(t) active, I2(t) through I5(t) are zero in figures 3 and 5, and are a "fuzzy analog zero" in Figure 4). "τ" is the square wave "separation" time of the input (I(t)) signals. For example, analog square waves of 0.1 sec "high", 0.2 sec "low" were studied, and reported in Figure 4. Please see Nold and Canney (1989), and Nold, Canney, and Georgioupolous (1990) for a more detailed description of this study.
"CLEAN" NUMERICAL SIMULATION

<table>
<thead>
<tr>
<th></th>
<th>NONGATED</th>
<th>GATED</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u &lt; u_0$</td>
<td>checks</td>
<td>checks</td>
</tr>
<tr>
<td>$u = u_0$</td>
<td>checks</td>
<td></td>
</tr>
<tr>
<td>$u &gt; u_0$</td>
<td>&quot;Bottom out&quot; for all $T_s &gt; 1.8$ sec</td>
<td>checks</td>
</tr>
</tbody>
</table>

Figure 3. The numerical performance of the Outstar is, not surprisingly, identical to (or "checks" with) the theoretical predictions.

"T checks" indicates that the training mode for the conditions defined in figures 3-5, approach the theoretical performance predictions of Grossberg (1967, 1982). "R decays", indicates that the LTM weights gradually decay due to the accumulation of "offset error" around the Outstar feedback loops (see figures 2, 4, and 5).

The "bottom out" condition for $u >> u_0$ discussed in figures 3-5, follows from (4) as:

Digital : $z(t) = e^{-ut} = e^{-50(1.8)} < 10^{-38}$
(VAX "underflow" limit)

Analog : $z(t) = e^{-ut} = e^{-50(0.2)} < S/N floor.$
(Signal/Noise floor of SIMSTAR analog circuit; dynamic range of approximately 60dB.)

Figure 3 describes the performance of the "clean" numerical simulation of both the nongated and gated systems studied. This figure (aside from the "bottom out") describes a performance identical to the Outstar learning theorems in Grossberg (1967, 1982).

Figure 4 outlines the difficulties encountered in the analog circuit performance. The circuit was scaled about the nominal value of $u_0 = 10$. This is the value the analog scaling was adjusted to. Scaling's importance can be appreciated when noting that the ratios in the dynamic ranges of the numerical versus the analog systems: $10^6$ versus $10^3$ respectively. The small $u$ rail off's were related to this limited dynamic range. When the learning rate was sufficiently "slowed down" to some scale dependent value of $u << u_0$, the Outstar apparently began to "chase" the offsets into saturation, or "rail off".

Figure 5 discusses the digital simulation of the e" model introduced for the multiplier offsets. Note this model fairly well describes the measured instabilities outlined in figure 4. The difference in the small $u$ behavior is still under investigation. The digital model, for $u << u_0$, approaches the rails at a slower, more "linear" rate compared to the analog.

ANALOG IMPLEMENTATION

<table>
<thead>
<tr>
<th></th>
<th>NONGATED</th>
<th>GATED</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u &lt; u_0$</td>
<td>T &quot;rains off&quot; or saturates</td>
<td>T &quot;rains off&quot; or saturates</td>
</tr>
<tr>
<td>$u = u_0$</td>
<td>T checks, $R$ decays</td>
<td>T checks, $R$ decays, but less rapidly</td>
</tr>
<tr>
<td>$u &gt; u_0$</td>
<td>&quot;Bottom out&quot; for all $T_s &gt; 0.2$ sec</td>
<td>T checks, $R$ decays</td>
</tr>
</tbody>
</table>

Figure 4. Note the performance difference compared to the numerical simulation results of Figure 3.

NUMERICAL "E" MODEL SIMULATION

<table>
<thead>
<tr>
<th></th>
<th>NONGATED</th>
<th>GATED</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u &lt;&lt; u_0$</td>
<td>T &quot;linearly&quot; tracks offsets towards &quot;rail&quot;</td>
<td>T &quot;linearly&quot; tracks offsets towards &quot;rail&quot;</td>
</tr>
<tr>
<td>$u = u_0$</td>
<td>T checks, $R$ decays</td>
<td>T checks, $R$ decays, but less rapidly</td>
</tr>
<tr>
<td>$u &gt; u_0$</td>
<td>&quot;Bottom out&quot; for all $T_s &gt; 1.8$ sec</td>
<td>T checks, $R$ decays</td>
</tr>
</tbody>
</table>

Figure 5. The simulated "e" model is seen to closely approximate the analog implementation performance noted in Figure 4 above.

One might view Figures 3-5 as pointing out two different levels of "machine limitation", when attempting to invoke the Outstar "plant", or "functional operations", or "learning law" model. The first level is commonly shared by both the digital and analog "substrates" (meaning: "supporting medium", mechanism, or "machine"). This "machine limitation" is that of dynamic range. This limiting level is noted in the "bottom out" (underflow), and the "rail off", or saturation (underflow, or overflow), which all refer to dynamic range limitations. Note that the "gated" (5) Outstar reduces much of the limitations of the "nongated" (4) system (see figures 3-5, for $u >> u_0$, and figures 4 and 5 for the $u = u_0$ case).
The second "machine limitation" level, peculiar only to the analog substrate, are the "analog multipliers", for which we have created the simple "e" model (see (6), (7), and (8), and Figure 5). Electronic analog multipliers are (and apparently always will be) incapable of performing a precisely "unique", or consistent mapping. This nonlinear circuit function always invokes a "small" random nonzero-mean (over finite time) offset, or bias in the product term (Sheingold 1976). This "e" bias term slowly wanders (relative to the faster LTM time constants), and is in general, absolutely unavoidable. Apparently, mother nature's view is that these offsets simply belong there!

CONCLUSION

The Outstar system has been shown to exhibit unstable LTM weight performance in the presence of arbitrarily small offsets in the multiplier product terms. Due to the classically understood unavoidable nature of these "e" offsets in all analog electronic systems, we conclude that this experimental result merits further theoretical consideration. One might hypothesize that these multiplicative offsets are somehow innate to complex physical (and perhaps even biological) nonlinear feedback systems.

Deeper theoretical considerations, perhaps via some sort of "canonical perturbation approach" (Tabor 1989), may possibly lead to a bridging between certain "neuro-physico-neuro-psycho" theories. Freeman (1987), lead by ingenious experimental measurement technique and interpretation, has developed a unique chaotic dynamic modeling approach. Grossberg (1982) lead by a brilliant "inductive/deductive" approach to a broad psychological data base, has developed a stable, sensitive, and adaptive learning theory. Noid, Canney, and Georgiopoulos (1990) speculate that a consideration of fundamental perturbations introduced by the simple process of multiplying two analog (physically continuous) variables, may help lead to an understanding of the yet undiscovered "homeostatic" mathematical principles of both "living", and "natural learning".

REFERENCES


List of Authors

Adams-Webber, J. R. 90
Ahlers, R. 49
Al-Khatib, L. 127
Anger, F. 274
Araya, C.L. 173
Aziz, A. R. A. 263
Barone, J. 177
Basehore, P. 280
Berlin, D. R. 164
Bezdek, J. C. 230
Biswas, G. 230
Biswas, P. 96
Bobbie, P. O. 23
Bolling, R. 60
Bowyer, K. M. 10, 60, 65, 70
Brown, F. M. 173
Brown, H. 289
Brown, H. K. 271
Burg, J. 258
Butler, R. M. 268
Cannon, R. 230
Carciofini, J. 13
Cheng, A. M. 202
Chung, H.-T. 145
Clark, D. 254
Clarke, T. L. 75
Colburn, T. 13
Cook, D. J. 28
Cross, D. 271
Derry, S. J. 240
Ernst, J. 53
Evans, R. 42
Fishman, M. 133
Ford, K. M. 90, 154, 214
Frederick, T. J. 159
Garcelon, J. H. 168
Gawronski, R. 274
Georgiou, M. 1, 284
Girle, R. 117
Goldgof, D. B. 10, 80, 85
Gonzalez, A. J. 189
Greer, S. C. 268
Hadlock, F. 133
Hall, L. O. 33, 202
Hawkes, L. W. 240
Heileman, G. H. 1, 284
Hoffman, R. 57
Holder, L. B. 149
Hughes, C. 258
Hundal, S. S. 173
Industrious, C. P. 33
Joshi, R. 122
Kambhamettu, C. 85
Kandel, A. 197, 254
Kegelmann, H. W. 240
Kelly, J. J. 214
Kendall, C. 230
Kim, L. 33
Kim, Y. G. 140
Kincaid, P. J. 194
Klose, G. 220
Kohout, L. J. 210
Kohout, L. J. 140
Koller, A. M. 42
Lafferty, L. 42
Lang, S. D. 194
Lange, D. F. 289
Leong, S. 250
Lim, H. 112
Mishra, S. K. 80
Morris, R. A. 127, 189, 245
Mosheili, J. M. 159, 258
Myler, H. R. 1
Nevill, G. E. 168
Novak, J. 90
Petty, M. D. 159
Pierce, E. A. 19
Pollack, R. B. 19
Raju, P. S. 6
Reed, J. 280
Rodriguez, R. 274
Sallam, M. 70
Sargeant, J. M. 107
Schneider, M. 112, 122, 164, 245
Schnitzius, M. 49
Schuerger, J. M. 107
Schumann 42
Schwartz, D. G. 145
Silverstein, E. E. 184
Simonian, R. 225
Simpson, K. M. 65
Smith, S. 197
Solano, J. L. 268
Stabile, I. 210
Stahl, H. 90
Sun, P. 184
Syu, I. 194
Tamir, D. E. 245
Taylor, G. 42
Thomas, M. 194
Tokuta, A. 85
Tran, D. 101
INDEX

2.5-D sketch..................................................65
3-D modeling..................................................10
3-D object representation..............................60, 70
action-based notation..................................85
Ada.............................................................225
genaggregation.................................................96
analogical planning........................................28
analogy...........................................................28
analytical solutions...........................................263
approximate reasoning.....................................145
aquifer parameters..........................................263
architecture.....................................................240
architectures for knowledge-based systems...........189
array manipulation............................................207
artificial neural network...................................284
aspect graph...................................................60, 70
assimilation theory.........................................90, 154
association.....................................................96
ATN grammar....................................................194
attributes.......................................................96
authoring tools...............................................107
automated knowledge acquisition.........................53, 107
automated logical inference................................210
automated theorem proving................................140
automatic programming.....................................207
autonomous land vehicle....................................159
back-propagation algorithm................................284
backward chaining..........................................112
belief revision.................................................154
bit matrix.......................................................112
bit-slice IC implementation...............................280
blackboard architecture.....................................225
cognitive modeling..........................................38
commonsense modeling.......................................220
computer vision.............................................60, 65, 80
concept maps..................................................90, 101
concurrent processes.........................................1
Connection Machine..........................................28
consistent labeling..........................................258
constraint propagation......................................240
constraint satisfaction......................................184
constraints...................................................19, 184, 258
constructive solid geometry system......................60
constructivist epistemology................................90, 154
deduction........................................................173
deduction tree search.......................................13
deductive synthesis.........................................207
Dempster-Shafer..............................................230
dendritic tree..................................................274
design theory...................................................168

313
<table>
<thead>
<tr>
<th>Topic</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>knowledge maintenance</td>
<td>149</td>
</tr>
<tr>
<td>knowledge representation</td>
<td>122, 127, 149, 173, 184, 220, 245</td>
</tr>
<tr>
<td>knowledge sources</td>
<td>38</td>
</tr>
<tr>
<td>knowledge-based systems</td>
<td>23, 184, 250, 268</td>
</tr>
<tr>
<td>knowledge-editor</td>
<td>96</td>
</tr>
<tr>
<td>learning</td>
<td>28, 164, 284</td>
</tr>
<tr>
<td>learning function</td>
<td>164</td>
</tr>
<tr>
<td>linear conformal motion</td>
<td>80</td>
</tr>
<tr>
<td>linguistic information</td>
<td>145</td>
</tr>
<tr>
<td>linguistic variables</td>
<td>145</td>
</tr>
<tr>
<td>local area networks</td>
<td>271</td>
</tr>
<tr>
<td>logic programming</td>
<td>13, 145, 258, 268</td>
</tr>
<tr>
<td>LogLisp</td>
<td>13</td>
</tr>
<tr>
<td>lower structure</td>
<td>220</td>
</tr>
<tr>
<td>machine learning</td>
<td>149, 154, 159</td>
</tr>
<tr>
<td>machine vision</td>
<td>75</td>
</tr>
<tr>
<td>massively parallel architectures</td>
<td>28</td>
</tr>
<tr>
<td>means-ends analysis</td>
<td>149</td>
</tr>
<tr>
<td>medical application</td>
<td>210</td>
</tr>
<tr>
<td>modal logic</td>
<td>117, 173</td>
</tr>
<tr>
<td>model-based reasoning</td>
<td>214</td>
</tr>
<tr>
<td>modeling</td>
<td>38, 258</td>
</tr>
<tr>
<td>motion analysis</td>
<td>80</td>
</tr>
<tr>
<td>natural language processing</td>
<td>194</td>
</tr>
<tr>
<td>natural language translation</td>
<td>133</td>
</tr>
<tr>
<td>natural language understanding</td>
<td>292</td>
</tr>
<tr>
<td>network-independent design</td>
<td>280</td>
</tr>
<tr>
<td>neural networks</td>
<td>1, 6, 75, 168, 177, 263, 271</td>
</tr>
<tr>
<td>neurons</td>
<td>274</td>
</tr>
<tr>
<td>NEXPERT OBJECT</td>
<td>42</td>
</tr>
<tr>
<td>nonclassical logics</td>
<td>145</td>
</tr>
<tr>
<td>nonmonotonic logic</td>
<td>173</td>
</tr>
<tr>
<td>nonmonotonic reasoning</td>
<td>173</td>
</tr>
<tr>
<td>nonnormal logics</td>
<td>117</td>
</tr>
<tr>
<td>nonrigid models</td>
<td>70</td>
</tr>
<tr>
<td>nonrigid motion analysis</td>
<td>10, 80</td>
</tr>
<tr>
<td>numerical analysis</td>
<td>6</td>
</tr>
<tr>
<td>object-oriented knowledge-based systems</td>
<td>90</td>
</tr>
<tr>
<td>object-oriented programming</td>
<td>258</td>
</tr>
<tr>
<td>object-oriented development</td>
<td>225</td>
</tr>
<tr>
<td>object-oriented programming</td>
<td>1, 240</td>
</tr>
<tr>
<td>objects</td>
<td>96</td>
</tr>
<tr>
<td>oil exploration</td>
<td>230</td>
</tr>
<tr>
<td>ontology</td>
<td>220</td>
</tr>
<tr>
<td>paraconsistent logics</td>
<td>117</td>
</tr>
<tr>
<td>parallel computing</td>
<td>268</td>
</tr>
<tr>
<td>parallel distributed processing</td>
<td>284</td>
</tr>
<tr>
<td>parallel environments</td>
<td>23, 33</td>
</tr>
<tr>
<td>parallel processing</td>
<td>271, 292</td>
</tr>
<tr>
<td>participatory learning</td>
<td>154</td>
</tr>
<tr>
<td>pattern matching</td>
<td>159</td>
</tr>
<tr>
<td>Term</td>
<td>Page(s)</td>
</tr>
<tr>
<td>----------------------</td>
<td>---------</td>
</tr>
<tr>
<td>type-curve matching</td>
<td>263</td>
</tr>
<tr>
<td>upper structure</td>
<td>220</td>
</tr>
<tr>
<td>user/computer interface</td>
<td>49</td>
</tr>
<tr>
<td>utility problem</td>
<td>149</td>
</tr>
<tr>
<td>validation</td>
<td>197, 202, 295</td>
</tr>
<tr>
<td>validation tools</td>
<td>295</td>
</tr>
<tr>
<td>value-classes</td>
<td>96</td>
</tr>
</tbody>
</table>
flairs

florida artificial intelligence research symposium