J. Indian Inst. Sci., Nov.-Dec. 1987, 67, pp. 465-490.

An expert system shell for mixed initiative reasoning*†

GAUTAM BISWAS
Department of Computer Science, Box 1688, Station B, Vanderbilt University, Nashville, TN 37235, U.S.A.

AND

TEJWANSH S. ANAND
Philips Laboratories, North American Philips Corporation, 345 Scarborough Road, Briarcliff Manor, NY 10520, U.S.A.

Abstract

This paper presents a general purpose expert system shell that incorporates mixed-initiative reasoning and the Dempster-Shafer (D-S) scheme of evidence combination for inexact reasoning. Domain knowledge is stored in the form of rules with associated belief values defined in the D-S framework. To structure the consultation process, the knowledge base is implemented as a partitioned rule base. The reasoning component uses a combination of forward and backward inferencing mechanisms, and controls a mixed initiative interaction with the user. To provide a suitable framework for performing the D-S computations and to achieve efficiency in propagating belief values during the chaining process, the rule base is compiled into a network. A rule editor has also been designed to facilitate knowledge-base construction.

Key words: Knowledge-based systems, inexact reasoning, Dempster-Shafer scheme, mixed initiative reasoning.

1. Introduction

Expert systems are a class of computer programs that emulate the problem-solving skills of human experts in specialized domains. They differ from conventional programs which use fixed algorithms for manipulating data, in that they can piece together a large amount of fragmentary knowledge in the form of facts and judgemental rules used by human experts, to solve a given problem[1]. They address problems normally thought to require the specialized knowledge of human experts for their solution. Expert systems have been successfully developed in fields such as medical diagnosis[2], equipment failure diagnosis[3], computer configuration[4], mineral exploration[5], and chemical data interpretation[6].

Earlier expert systems, such as MYCIN[2] and PROSPECTOR[5] for the most part were developed from scratch using dialects of Lisp (*e.g.*, MYCIN was developed in the Interlisp environment). Developing an expert system from scratch is an enormous task,

*First presented at the Platinum Jubilee Conference on Systems and Signal Processing held at the Indian Institute of Science, Bangalore, India, during December 11-13, 1986.

†This work was originally conducted by the authors at the Department of Computer Science, University of South Carolina, Columbia, S.C. 29208.

partly because the developers have to split their efforts between two major tasks: (i) knowledge acquisition from experts, and (ii) designing knowledge manipulation procedures. However, the development of earlier systems indicated that the inferencing and knowledge representation components could be separated from the domain-specific knowledge base and applied to a completely different task (For example, SACON[7], a system for structural analysis, and a number of other systems[2] were developed using EMYCIN (essential MYCIN, *i.e.*, MYCIN without its domain-specific knowledge base). Such tools, or shells as they are sometimes called, facilitate rapid development of the initial prototype of a knowledge-based system.

Currently, a number of software tools are available for knowledge engineering. They fall into two categories: (i) skeletal systems extracted from previously built expert systems, and (ii) general-purpose languages developed specifically for the knowledge engineering task. In a skeletal system, all domain-specific knowledge is stored in the knowledge base in a predefined format, and operated on by an inference engine that is very similar to the system from which it is extracted. Examples of skeletal systems are EMYCIN[8] (derived from MYCIN), KAS[9] (derived from PROSPECTOR) and EXPERT[10] (derived from CASNET). These tools are suitable for application domains where the problem-solving structure and strategies are similar to the original system from which these tools have been extracted. General-purpose languages are less constrained than skeletal systems, and provide flexible programming environments because they are not closely tied to any particular framework and allow for a wider variety of control structures. Thus, they may be efficiently applied to a broader range of tasks, although the process of developing the system may be more complicated. ROSIE[11], OPS5[12], RLL[9] and HEARSAY-III[13] are examples of general-purpose programming languages developed specifically for knowledge-based system development.

Most tools listed use the rule-based approach for knowledge representation, although the format and structure of the rules differ. For example, EMYCIN uses object-attribute-value triples, OPS5 uses attribute-value pairs, and ROSIE uses deductive or logic rules. In addition to rules, KAS uses semantic networks to represent classificatory information in the domain, and ROSIE uses a declarative data base to store simple English-like assertions. Uncertainty is also handled in a variety of ways: ROSIE and OPS5 have no in-built mechanisms for inexact reasoning. KAS uses an approximation of the Bayesian conditional probability scheme, whereas EMYCIN and EXPERT use *ad hoc* mechanisms. The inferencing mechanisms use either forward- or backward-chaining strategies with the exception of KAS which uses a combined forward-backward strategy. All the tools provide some facilities for knowledge acquisition in the form of rule editors, and debugging and testing aids in the form of trace functions. However, none provide more sophisticated debugging aids such as the ones developed in TEIRESIAS[14]. Some of the newer tools, such as ART, KEE and LOOPS[1] are flexible hybrid tools, *i.e.*, they can be used to represent knowledge in several different ways, and can be applied to several different paradigms. Table I summarizes the salient characteristics of a few of the expert system shells that are in use now-a-days.

In this paper, we present MIDST (a Mixed Inferencing Dempster-Shafer Tool) that incorporates inexact reasoning mechanisms based on the Dempster-Shafer evidence

Table I
Characterization of some expert system tools

Name	EXPERT	EMYCIN	KAS	OPS5	ROSIE
Knowledge representation	Production rules operating on propositions.	Production rules operating on associative (object-attribute-value) triples.	Production rules that link antecedents to consequents. Partitioned semantic networks represent taxanomical information bearing on antecedent and consequent situations.	Production operating on lists of attribute-value pairs.	Set of propositions acted upon by rules. General *n*-ary relations permitted. English-like syntax.
Inexact reasoning	Belief values in the interval $[-1, 1]$ attached to facts. Values used as thresholds; no combination formula.	Certainty factor; a normalized probability $\in[-1, 1]$ attached with every triple. *Ad hoc* combination formula.	Probabilistic Bayesian framework. Two likelihood ratios, measures the degree of sufficiency and necessity associated with each rule.	No built-in mechanism.	No built-in mechanism.
Inferencing strategy	Questionnaire-driven (designer specified order for data collection). System forward chains once all data are gathered. Rules fired in prespecified order.	Backward chaining, initial goal is to determine the value of a top-level goal-attribute.	Mixed-initiative control, achieved by combining forward and backward chaining.	Forward chaining, recognize-act cycle. Two conflict resolution strategies. Incorporates an efficient pattern match algorithm.	Forward or backward chaining. Powerful pattern matching strategies.
Knowledge editor	No specific aids or editor for knowledge base construction.	High-level knowledge-based editor checks syntactic validity, contradiction and subsumption.	Knowledge-based editor that operates directly on network structures.	No specific aids or editor for knowledge base construction.	No knowledge-based editor. Hard to add or modify rules. Can use any text editor.
Debugging aids	Trace facilities. Statistical functions that keep tabs on rule usage.	Tracing facilities, HOW and WHY explanations.	Immediate feedback on consequences of changes to knowledge base.	Trace and break facilities to track intermediate steps in reasoning.	Trace operations. Helpful error messages and error recovery.
Application paradigm	Classification (diagnosis/prescription) problems.	Consultation programs for diagnosis.	Consultation environments.	General programming environment.	Constructing knowledge-based systems.
Environment	FORTRAN	INTERLISP	MACLISP	LISP	INTERLISP

combination scheme. Our discussion will concentrate more on the design and implementation of the rule network, and the inferencing mechanism that combines forward- and backward-chaining procedures to achieve a mixed-initiative dialogue with the users of the system.

Section 2 of the paper briefly reviews and compares existing numerical schemes for inexact reasoning. Section 3 presents the MIDST system architecture, and then discusses in detail the inferencing mechanisms along with the underlying knowledge base structures. Section 4 briefly outlines some of the knowledge-base construction aids available in MIDST, and then discusses the framework for acquiring belief functions associated with the rules of the knowledge base. Section 5 presents the conclusions and directions for future work.

2. Inexact reasoning

In complex domains, efficient problem solving often requires the use of judgemental knowledge provided by human experts, which can often be conveniently represented as if-then rules. The uncertainty or judgemental nature of the expert's knowledge is expressed by qualifying conclusions with terms such as *likely*, *suggests*, *lends credence to*, etc. Therefore, it becomes necessary for the knowledge engineer designing the system to represent and reason with this uncertainty in the problem-solving model. At least four separate causes for uncertainty can be identified. The first, described above, is due to the inherent uncertainty in expert-supplied heuristics. The second cause is related to the reliability of information, *i.e.*, the facts or data required for solving the problem may be imprecise. The third cause for uncertainty in conclusions is that they are often based on incomplete information. A fourth cause for uncertainty arises when problem-solving information accumulated from multiple sources turns out to be contradictory. It is this pervasive imprecision and uncertainty in the real world that requires the adoption of inexact reasoning mechanisms in computer-aided decision making.

One approach to solving this problem is by numeric modeling. This involves assigning numerical values to the linguistic terms used to qualify the conclusions made in rules; *e.g.*, the value 0.6 may represent the phrase *suggests*. The problem now reduces to interpreting these numbers in a suitable framework, and then defining appropriate functions for combination of evidence in that framework.

The traditional approach to uncertain decision making has been the use of probability theory and Bayes combination formula for conditional probabilities; well-known examples of systems that employ Bayesian reasoning are PIP[15], CADUCEUS[16], and PROSPECTOR[5]. These systems interpret belief values as conditional probabilities, *i.e.*, the probability of observing the hypothesis on the right-hand side of a rule given the evidence described on the left-hand side of the same rule. When multiple evidence supports the same hypothesis the overall probability or likelihood of the hypothesis is computed by applying Bayes' formula. However, there are a number of problems associated with the practical implementation of Bayesian schemes. Szolovits and

Pauker[17] have shown that a direct implementation of Bayes' formula leads to a very large and unmanageable database even if simplifying assumptions such as independence of individual observations are made even though they are hard to justify physically. Again, in the probabilistic framework, the probability that a given hypothesis h_i is true, $P(h_i)$, and the probability that h_i is false, $P\ (-h_i)$, are linked by the formula:

$$P(h_i) + P(-h_i) = 1. \tag{1}$$

Thus, if $P(h_i) = 0$, we must have $P(\sim h_i) = 1$. But when we have no *a priori* knowledge regarding truth or falsity of h_i, ignorance is difficult to represent. To work around this problem, the assumption $P(h_i) = P(\sim h_i) = 1/2$, is often made in order to represent a state of ignorance. However, when the number of possibilities increase to more than two, Shafer[18] has demonstrated, by simple examples, that the technique of assigning equal probabilities can produce counter-intuitive results. Because of the distortions it imposes on the problem, and because of its enormous data requirements, pure probabilistic schemes tend to be successful only in small, well-constrained problem domains[17]. Most real-life problems that involve complex decision making suffer from insufficient data and imperfect knowledge, so a rigorous probabilistic analysis is not possible. To avoid some of the problems that come along with the application of a formal probabilistic framework, *ad hoc* functions for the combinations of evidence have been proposed. A well-known example is MYCIN[2], a diagnostic expert system for selecting antibiotic therapy for bacteremia. The system uses heuristic functions based on confirmation theory or subjective probability, and the mechanism for evidence combination conceived on purely intuitive grounds has proved to be a good approximation to the intuitive methods used by doctors. Although this scheme was originally proposed as an alternative to probability theory, Adams[19] has shown that combining functions can be derived from probability theory with the assumption of statistical independence.

The drawbacks of the Bayesian and *ad hoc* schemes like the one used in MYCIN have drawn attention to the Dempster-Shafer (D-S) theory in decision-making applications[20]. The key advantages of this scheme are its abilities to allow belief to be assigned to subsets of hypotheses from the space of possibilities (as opposed to singleton hypotheses in the Bayesian framework), effectively represent the concept of ignorance in the reasoning process, and model the narrowing of the hypotheses set with the accumulation of evidence. This provides a better model for the expert reasoning process. Belief functions and their combining rule in the D-S theory are well suited to represent incremental accumulation of evidence and the results of its aggregation. Shafer[18] has shown that the D-S approach includes the Bayesian and MYCIN evidence combination functions as special cases. In addition, it avoids the probabilistic restriction of eqn (1). The definition of a belief function, and the mathematical formulation of the evidence combination scheme are presented next.

2.1 The Dempster-Shafer approach to inexact reasoning

The following discussion follows the notation used in Shafer[18] and Gordon & Shortliffe[20]. The D-S formulation is based on a frame of discernment, Θ, a set of propositions or hypotheses about the exclusive and exhaustive possibilities in the domain under consideration. The notation 2^Θ is used to denote the set of all subsets of Θ. Further discussion is based on two concepts that can be adopted for the representation of evidence: the *measure of belief* committed exactly to a subset A of Θ (*i.e.*, $A \in 2^\Theta$) and the *total belief* committed to A. Exact belief relates to the situation where an observed evidence implies the subset of hypotheses, but this evidence does not provide any further discriminating evidence between individual hypotheses in A. A function $m{:}2^\Theta \rightarrow [0,1]$ is called a *basic probability assignment* (bpa) whenever:

$$m(\Phi) = 0, \text{ and} \tag{2a}$$

$$\sum_{A \subset \Theta} m(A) \leq 1 \tag{2b}$$

where 2^Θ is the set of all subsets of Θ. The quantity $m(A)$ represents the measure of belief that is committed *exactly* to A. In other words, a bpa represents the support a piece of evidence provides to subsets of Θ. Condition (2a) reflects the fact that no belief ought to be committed to Φ, the null hypothesis, while (2b) states the convention that one's total belief has to sum to less than or equal to one. The measure of *total* belief committed to a subset A is defined as:

$$Bel(A) = \sum_{B \subseteq A} m(B) \ , \tag{3}$$

where the summation is conducted over all B that are subsets of A. A function Bel: $2^\Theta \rightarrow [0,1]$ is called a *belief function* over Θ, if it is given by (3) for some basic probability assignment m. If (2b) sums to less than 1, then $(1 - \sum_{A \subset \Theta} m(A))$ defines a measure of ignorance, denoted by $m(\Theta)$. In other words, $m(\Theta)$ is the extent to which the observations provide no discriminating evidence among the hypothesis in the frame of discernment, Θ.

Judgemental rules provided by experts basically represent individual pieces of evidence that imply subsets of hypotheses with confidence values that correspond to measures of exact belief (details of rule structure are given in Section 3.1). Corresponding to two different pieces of evidence e_1 and e_2 with bpas m_1 and m_2, respectively, over the same frame of discernment; Dempster's rule of orthogonal products is applied to combine the effects of observing the two pieces of evidence and compute a new bpa, m, that is given by:

$$m(C_k) = \frac{\sum_{A_i \cap B_j = C_k} m_1(A_i) m_2(B_j)}{1 - \sum_{A_i \cap B_j = \varnothing} m_1(A_i) m_2(B_j)}$$

where A_i represents hypotheses subsets that are supported by e_1, B_j represents hypotheses subsets supported by e_2, and C_k represents the hypotheses subsets that are supported by the observation of both e_1 and e_2. The denominator is a normalizing factor to ensure that no belief is committed to the null hypothesis (condition 2a). More detailed discussions on the Dempster-Shafer theory of evidence combination appear in Shafer's work[18].

3. The MIDST system

As discussed earlier, a number of expert system development tools are available now a days. A large number of these tools require the encoding of domain knowledge in the form of pattern-action (or, antecedent-consequent) rules. Uncertainty representation in the form of numbers in the interval [0,1] or [−1,1], and inexact reasoning methods have also been incorporated into some of these systems. Current schemes are based on approximate Bayesian methods (*e.g.*, PROSPECTOR), or *ad hoc* schemes (*e.g.*, MYCIN). Some others, such as FLOPS[21], use fuzzy reasoning techniques. However, to date there are very few systems based on the D-S scheme in spite of its advantages discussed in Section 2. Some of these advantages were demonstrated in a system called OASES, developed for trouble-shooting production processes[22,23]. In the rest of this section and the next, we discuss the design and implementation of MIDST, our expert system shell extracted from OASES.

Figure 1 illustrates the basic components of the MIDST system. The *system designer* or the *expert* interacts through the *system designer interface* to create a knowledge base for problem solving.. This interface provides two main functions. *Via* the rule editor the system designer builds the domain knowledge base as a partitioned rule base. Facilities are provided for creating, deleting and modifying partitions, and then creating, deleting and modifying individual rules within partitions. The second function of the interface is to provide debugging feedback to the system designer and expert during the system development process. During system development after the system designer enters rules for a partition or makes changes to them, the system is designed to analyse the rules and determine if any rule is missing or is incomplete. We are still in the process of implementing this functionality of the system along with other debugging techniques, and, therefore, do not discuss them in this paper.

The *rule editor* performs the task of converting rules entered in an English-like format into the Internal Lisp format required by MIDST. Uncertainty in the rules is represented as exact belief functions in the D-S framework, and are extracted from the domain expert employing a scheme we describe in Section 4. Rules are stored in the *rule base*, and associated queries are stored in the *query database*. The *network compiler* compiles the rules into a *rule network* which makes the inferencing process efficient. The compilation of rules into a network is an expensive process, and is done off-line.

The task of interpreting the domain knowledge is performed by the *inference engine* which is the heart of the MIDST system. MIDST incorporates a combined forward- and backward-control structure in its reasoning mechanism. This allows the system to have

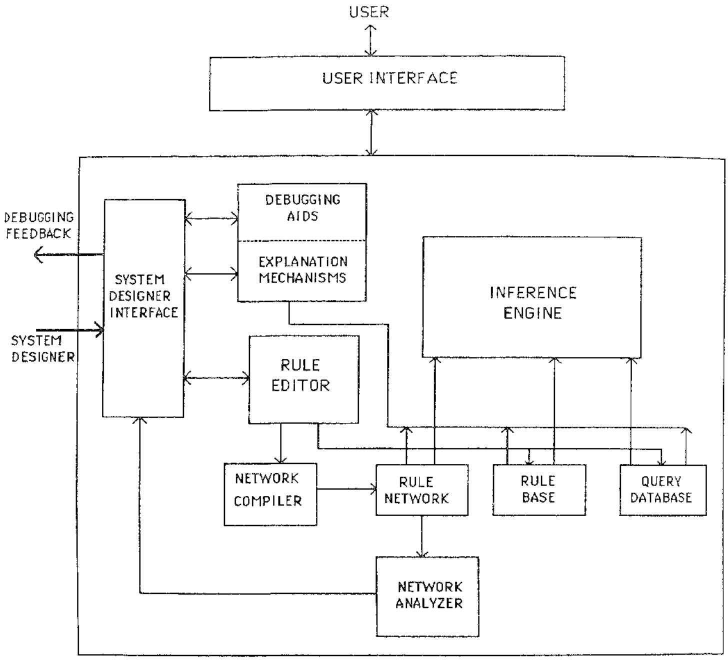

FIG. 1. MIDST system architecture.

mixed-initiative control. A typical session starts with the system allowing the user to volunteer information that might be relevant to the problem. This information is used by the system to derive intermediate or partial conclusions. The system then assumes control, and what typically follows is a cycle of questions and answers in which the inference engine selects a relevant piece of evidence to query the user about. Instead of answering a question the user can take the initiative any time and volunteer additional evidence, seek clarification, or request an explanation. This latter request is handled by the *explanation mechanism* sub-system and takes one of two forms: (i) an explanation of the reason for that particular question, or (ii) a summary of the principal conclusions of the system at that point in the consultation. This task is essentially performed by examining the history of the consultation. The ability of the system to examine the rule network and produce explanations is quite important as it makes the inferencing mechanism transparent to the user. Evidence combination is based on Dempster's combining formula (eqn 4) presented in Section 2. In the rest of this section, we discuss the structure of the knowledge base, the rule format, the design and implementation of the rule network and the reasoning strategy employed in MIDST.

3.1 Knowledge-base structure

The knowledge base of MIDST is formulated as a partitioned rule base. The structure of the rules (*i.e.*, the rule language) as well as the partitions are designed to facilitate the representation of domain knowledge, and incorporate uncertainty in terms of belief functions in the D-S framework.

A rule links a pattern on its left hand side (LHS) to one or more conclusions and/or a sequence of actions on its right hand side (RHS), *i.e.*, the LHS pattern represents relevant evidence for the conclusions on the RHS. Single pieces of evidence are represented as attribute-value pairs, and, in general, an LHS pattern is a conjunction of pieces of evidence, (*e.g.*, [(<*process type*> <*continuous flow*>) & (<*problem occurrence*> <*continual*>) & (<*insufficient capacity*> <*late in process flow*>)]). Actually, evidence in the LHS pattern of a rule can be one of two types:

(i) askable, corresponding to information that can be obtained by directly querying the user (therefore, they have expert-supplied queries associated with them), or
(ii) verifiable, corresponding to evidence obtained from rule firings. (They represent intermediate conclusions in the chaining process).

Each conclusion on the RHS is a disjunctive set of hypotheses, where an individual hypothesis is represented as an attribute-value pair. In addition to conclusions, the system designer may also specify a sequence of forward-chaining actions on the RHS of a rule. These actions are usually Lisp functions which permit the system designer to perform additional computations, or incorporate overriding control constraints.

To accommodate uncertainty in the rule structure, the shell associates an expert supplied belief value with every conclusion on the RHS. Belief values are modeled as a bpa function in the D-S framework. Note that belief values may also be associated with individual pieces of evidence in the LHS pattern. They may be derived from user input for askable patterns or computed values for verifiable patterns. *BF* is a Lisp function that computes the overall belief value for the LHS pattern of a rule. If multiple pieces of evidence are involved, the belief value associated with the LHS pattern is the *minimum* of the belief values of each piece of evidence on the LHS of the rule.

A rule from the OASES[22,23] system illustrates the rule structure described above:

[(<*process-type*> <*continuous flow*>) bf_a and
(<*insufficient capacity*> <*late in process flow*>) bf_b] →
{[(<*cause*> <*process design*>) 0.5]
[(<*cause*> <(*raw materials, technology, maintenance*)> 0.3]}.

The expert may also supply evidence that negates the belief in a conclusion. For example, in the OASES framework a rule stated in English is:

if (*continuous flow fiberglass manufacturing process*) & (*molten glass viscosity is not nominal*) & (*all ingredient compaction ratios are within limits*)
then (*rule out bin level fluctuations as the cause for the raw material sourcing problem*).

Such heuristic rules enable the expert to apply the process of elimination in the diagnostic process. In the D-S framework, evidence against a hypothesis is treated as evidence in favor of the negation of the hypothesis in the set theoretic sense. Therefore, if $\Theta = \{$*bin level fluctuations*, *inconsistency of raw materials*, *post-scale contamination*$\}$ then the above rule translates to:

if (*continuous flow fiberglass manufacturing process*) &
(*molten glass viscosity is not nominal*) &
(*all ingredient compaction ratios are within limits*)

then (*inconsistency of raw materials or post-scale contamination is the cause for the raw material sourcing problem*).

Facilities are provided for *partitioning* the knowledge base into separate chunks or units. This may facilitate the modeling of the expert's reasoning process. Conceptually, partitions represent the breakdown of a complex problem-solving process into a sequence of component subproblems. The division of the knowledge base into partitions is an attempt to streamline and make efficient the inferencing mechanism by imposing a structure on the problem-solving process. The partitioning approach also provides an efficient way to model the expert's reasoning process. As a result, the knowledge acquisition process can be made more structured, thus making a little simpler, an otherwise difficult task. Another benefit of partitioning, discussed later, is that it makes the D-S scheme easier to implement and computationally more efficient.

An important feature of the expert system shell is the compilation of rules into a network that makes the backward-chaining phase of the inferencing more efficient, and provides a convenient means for implementing the chaining process in the D-S framework. This compilation is done off-line and the resultant network is stored as a multi-linked list. The rule network is described in greater detail next.

3.2 *The rule network compiler*

The MIDST rule compiler takes advantage of knowledge available from a static analysis of the rule set to produce a more efficient representation for computational purposes. Specifically, it constructs a Lisp structure that embodies a block of rules that bear on the same diagnostic conclusion.

3.2.1 *The rule network design*

The rule network represents a compilation of individual rules, and links conclusions to relevant evidence. Each sequential partition is compiled into a disjoint rule network. For example, fig. 2(b) shows a portion of the rule network corresponding to rules in XX^{24},

```
(rule03
 (((<dist> <less_equal_200>)) BF)
 ((((<site of play> <shelf>)
   (<site of play> <margin>)) 0.8)) )

(rule04
 (((<dist> <greater_200>)) BF)
 ((((<site of play> <craton>)) 0.6)) )

(rule06
 (((<move> <seaward>)
   (<beds_dip> <seaward>)
   (<beds_deepen> <seaward>)
   (<abrupt_change> <no>)) BF)
 ((<site of play> <margin>)) 0.7)) )

(rule18
 (((<sed_finer> <seaward>)) BF)
 ((((<beds_deepen> <seaward>)) 0.7)) )

(rule19
 (((<sed_finer> <landward>)) BF)
 ((((<beds_deepen> <landward>)) 0.7)) )

(rule20
 (((<sed_homogeneous> <seaward>)) BF)
 ((((<beds_deepen> <seaward>)) 0.7)) )

(rule21
 (((<fauna_deepens> <seaward>)) BF)
 ((((<beds_deepen> <seaward>)) 0.7)) )

(rule22
 (((<reflectors_thin_&_dip> <seaward>)) BF)
 ((((<beds_dip> <seaward>)) 0.6)) )
```

FIG. 2a. Some of the XX rules.

that deal with the identification of the site of a hydrocarbon play. The rules are listed in fig. 2(a). Both conclusions and evidence are represented in the same conceptual framework: attribute-value pairs. They form the nodes of the network. Rules relate evidence patterns to conclusions, and appear as links in the network. Links have weights associated with them. These weights are directly dependent on the amount of belief that the evidence pattern provides for the particular conclusion it is linked to. Final conclusions represent the top layer of nodes in the rule network. In fig. 2(b), the top layer of the network represents the possible values of the site of a hydrocarbon play:

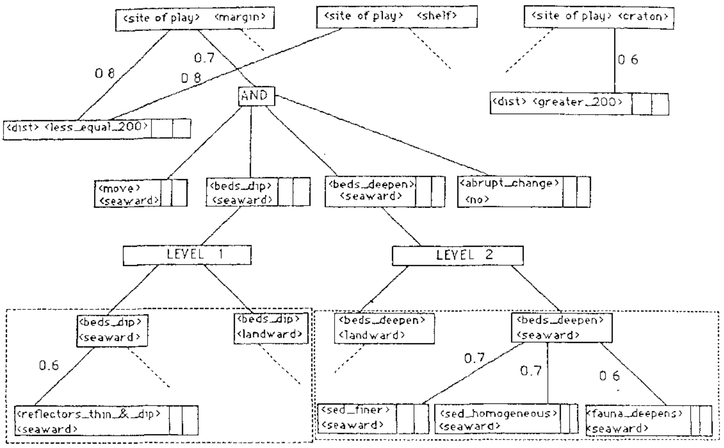

FIG. 2b. Section of the rule network for XX.

within the craton, on the continental shelf, and on the oceanic margin. In addition, there are two other kinds of *dummy* nodes. The first is an *AND* node: a conclusion that depends on a conjunction of evidences is linked to the evidence-nodes through this kind of node. The second is a *level* node. Level nodes link two hypotheses spaces. Conceptually, a hypothesis space is made up from the set of rules that verify the same attribute. It should be noted that the conclusions in a hypothesis space many be final or intermediate, and evidence may either be askable or verifiable. In fig. 2(b), the nodes enclosed in the dashed boxes represent hypotheses spaces. Nodes corresponding to verifiable evidence patterns are linked to level. Evidence nodes corresponding to the same attribute converge on to the same level-node. For example, in fig. 2(b) $<$ beds_ deepen $>$ is a verifiable attribute, and, therefore, all evidence nodes that support the values of this attribute (*seaward* or *landward*) are linked to the same level-node. This level node is also linked to the hypothesis space where a value for this attribute may be derived. Thus, the overall structure of the rule network is that of hypotheses spaces linked to each other through level-nodes.

In inferencing terminology, verifiable pieces of evidence represent intermediate conclusions and their presence leads to chaining, or reasoning at multiple levels. To handle chaining in the D-S framework each hypothesis space defines a separate frame of discernment (Θ). This approach closely mirrors the method suggested by Gordon and Shortliffe for implementing MYCIN[20] in this framework. The conceptual structures in the knowledge base are the attribute-value pairs, and each verifiable attribute defines a frame of discernment extending over the possible values of that attribute. An advantage of such an implementation is that it maintains mutual exclusiveness of hypotheses and independence of evidence, a requirement in the D-S framework.

A shortcoming of the current network implementation is that it does not completely capture all the information contained in the rules. For example, the network does not provide features to faithfully represent rules that have evidence-supporting multiple hypotheses, or evidence that supports a set of hypotheses rather than a singleton hypothesis. As a result, when the pattern on the LHS of a rule is instantiated, the actual rule is invoked to generate new hypotheses and compute the new belief function. In future, we will define additional dummy nodes to solve the above representation problem. The network would then provide effective speed up both in the forward- and backward-chaining modes of operation. However, in our current research the emphasis was on developing efficient backward chaining and query-selection mechanisms, and the current network configuration definitely achieves these objecives.

In summary, some of the salient features of the network implementation are:

(i) the presence of bidirectional links which permit easy traversal along the network structure, and

(ii) a table look-up mechanism that allows identification of nodes corresponding to an attribute. This allows a single-step access to any node in the network.

The utilization of the network for efficient query and rule selection during backward chaining, and propagation of belief values between hypotheses spaces is further elaborated in the next section.

3.3 *Inferencing/Control structures*

The inferencing mechanism has four main components: the evidence combination scheme, the procedure for selecting the top-ranked hypothesis, the query-selection mechanism that directs the user-system dialogue based on the top-ranked hypothesis, and the top-level controller that is related to the selection of partitions within the rule base. The system adopts a mixed-initiative form of control. Initially, the user may provide facts and evidence that he/she considers relevant to the problem. The system forward chains on this evidence, establishes intermediate conclusions, and then ranks goal hypotheses based on some criteria, *e.g*, the belief values associated with each goal hypothesis. The D-S scheme is used to update the belief values of hypotheses depending on the evidence provided by the user. MIDST then goes into the backward-chaining mode, identifies the top-ranked hypothesis and then the best question to ask the user to try and establish this hypothesis. If the user considers the current query to be irrelevant, he/she may provide additional facts and evidence, unrelated to the query, which switches the system back into the forward-chaining mode. This approach illustrates mixed-initiative control.

The overall flow of control for the inferencing mechanism is shown in fig. 3. The user interface is directed by the ASKQ routine. In the initial step, the system goes through questions in the top level partition (the partition that has an initial predetermined sequence of queries similar to the ASKFIRST queries in MYCIN[2]), and the user responses are converted into appropriate evidence patterns and stored in working memory. Depending on the responses, the partition controller transfers control to an

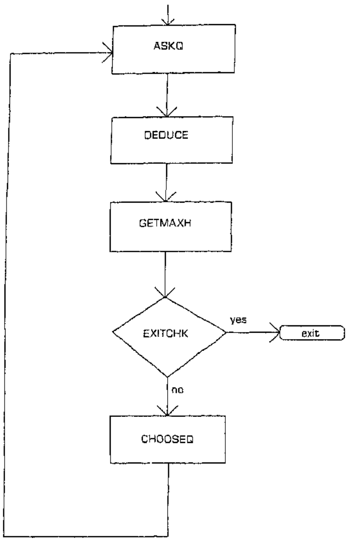

FIG. 3. The inference control structure.

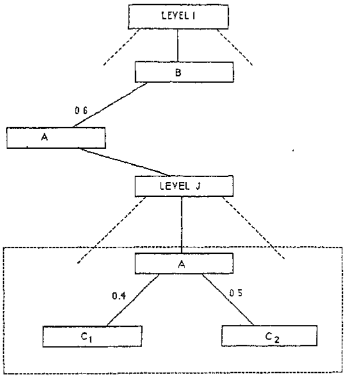

FIG. 4. Simple rule network.

appropriate partition. Forward chaining and evidence combination is performed by the DEDUCE function. This establishes intermediate conclusions based on which the final goal hypotheses are ranked. The function GETMAXH is invoked to pick the leading hypothesis. Next the backward-chaining process is initiated, and CHOOSEQ selects appropriate queries based on the top-ranked hypothesis. At each step, the exit condition is checked by an EXITCHK function. Before querying the user for more information, the system checks if, based on the current belief values, it can come to some definite conclusions. Instead of having a standard overall EXITCHK function, the shell allows the system designer to define EXITCHK routines for every partition. For example, the following conditions:

(i) the ignorance factor, m (Θ) is below a certain threshold,

(ii) the belief value committed to the top-ranked hypothesis exceeds a second threshold, and

(iii) the difference in belief values between the two top hypotheses exceeds a third threshold

were checked in the EXITCHK routine for the lower-level partitions of OASES[23] to establish the leading hypothesis beyond doubt. As a justification, we observe that a combination of the first two conditions ensure that further corroborating evidence would change the belief in the leading hypothesis only marginally, whereas the third condition ensures that the belief in the leading hypothesis is sufficiently greater than any of the other possibilities. Other EXITCHK conditions can be specified for individual levels.

3.3.1 *ASKQ*

As discussed earlier, user interactions are controlled by the ASKQ routine. This routine generates queries and invokes a separately defined routine independent of the shell (user interface in fig. 1) to interpret the user's response. Depending on the sophistication of the system interface this routine may incorporate some form of natural language processing. The routine returns an evidence pattern or a set of evidence patterns depending on the user's response. These patterns are stored in the MIDST system working memory. For example, in the OASES[23] system, the ASKQ routine incorporates a module GENEV which is based on a combined ATT parser-keyword matching approach. The user may provide a direct reply to the system query, or if he wishes to specify other relevant information, he may do so in his response.

3.3.2. *DEDUCE*

This is the forward-chaining routine. The initial algorithm adopted for DEDUCE can be described as follows:

1. HYPS ← list of hypotheses sets and associated belief values.
2. Match the evidence pattern generated by the user-supplied information with the current active rule set.
3. Find all rules that have satisfied left-hand sides (this is the instantiated set).
4. Trigger all rules in the instantiated set one by one, and update the belief values of the hypothesis in HYPS using Dempster's combination formula (The order in which rules are fired is not important).
5. Perform all forward-chaining actions associated with the rule being fired.

This algorithm reflects the traditional forward-chaining approach where the interpreter searches working memory, picks a set of rules that can be fired, makes a choice and fires a rule which updates the contents of working memory. This process is repeated till no rules can be fired. However, this straightforward approach could not be directly applied to MIDST since this would result in a large amount of extra computation, as is illustrated below. Consider a very simple example, with the following rules:

rule1: if A then B, 0.6,

rule2: if C_1 then A, 0.4,

rule3: if C_2, then A, 0.5.

The portion of the network corresponding to these three rules is shown in fig. 4. If working memory denotes that C_2 has been established with belief value = 1 then rule2 derives A with belief $= 0.5$, and this results in an update of belief values of all hypotheses in Level J frame of discernment. On propagation of belief in A, rule1 derives B, and, therefore, equation 4 can again be applied to update belief values of hypotheses in Level I frame of discernment. Now, if at some later time C_1 is established, rule2 will fire and the belief in A (and other Level J hypotheses) will be updated, and this change in belief will propagate to higher levels (say Level I), where belief values of all hypotheses will have to be recomputed. However, evaluation of equation 4 is computationally expensive, and it would be quite wasteful to repeat the entire computation every time the belief in an already established hypothesis or evidence is updated. A number of schemes have been proposed for making the belief function computations more efficient; the better known examples are schemes proposed by Barnett[25], an extension of Barnett's scheme proposed by Gordon and Shortliffe[20], and then a more generalized scheme proposed by Shenoy and Shafer[26].

But none of these directly address the situation described, where changes in belief values of hypotheses need to be propagated to higher levels so that belief values at these levels may be updated. However, this involves complete recomputation of all belief values at higher levels (which is computationally very expensive), whereas, ideally, the belief values should be updated by incremental recomputation. The options thus left open to us were, either to use some sort of approximation for incremental updating of belief values at higher levels, or use some other technique to handle this situation. We chose the latter. In order to prevent wasteful computations, belief values were not transmitted between levels, till the EXITCHK conditions, described in Section 3.3, were satisfied. Therefore, in the example above, the belief value for A would not be propagated to the next level till it was established that there would be no significant change in its belief value (either because of conditions (i) and (ii) in EXITCHK, or because the system knows of no other evidence, *i.e.*, rules, that can support the hypothesis). However, within a level, the DEDUCE function operates as described in the beginning of this section. In order to implement this scheme, local structures called hypothesis-bases are defined for each hypothesis space. These structures store intermediate beliefs in the hypotheses and once these beliefs achieve a 'sufficient' value, or all rules associated with this level have been exhausted, they are added on to the global evidence-base (working memory). The 'sufficient' criterion is defined by the EXITCHK conditions. The modified algorithm that incorporates this change is described below.

```
Do for all the newly acquired evidence patterns
 1. Determine the level in the network where the evidence pattern is applicable.
 2. Retrieve the local hypothesis-base of that level from the level-node in the network.
    HYPS ← list of hypotheses sets and associated belief values.
 3. Do until no more rules can be fired.
    a. Match the evidence pattern with the current atcive rule set.
    b. Find all rules that have satisfied left-hand sides (this is the instantiated set).
```

   c. Trigger all rules in the instantiated set one by one and update belief values of the hypotheses in HYPS using Dempster's combination formula (The order in which rules are fired is not important).
   d. Perform all forward-chaining actions associated with the rules being fired.
4. For every level if the local HYPS satisfies the EXITCHK conditions, add this HYPS to the global evidence-base and mark the corresponding level as *solved*.

3.3.3. *GETMAXH*

The function of the GETMAXH routine is to determine the most promising (*i.e.* the top ranked) hypothesis which the system then tries to establish by querying the user for more information. The procedure adopted for doing this is as follows:

1. Rank the hypothesis in the HYPS corresponding to the top-most level of reasoning in the current active partition.
   This ranking is done on the basis of the exact belief committed to the hypotheses.
2. Select hypothesis with maximum belief.
3. If this is a set, select hypothesis with maximum belief among the elements of the set.

3.3.4. *CHOOSEQ*

The query-selection mechanism is implemented by the routine CHOOSEQ. Given the leading hypothesis which the system is trying to verify, CHOOSEQ examines the network of compiled rules to find out which rule it needs to fire to increase belief in the leading hypothesis. The network, as discussed earlier, links hypotheses to relevant evidence patterns as described by the expert-supplied rules. If the most relevant pattern is askable, CHOOSEQ returns the corresponding query to ASKQ. On the other hand, if the pattern is verifiable, and the level associated with that attribute has not already been *solved* (in the sense of step 4 of the modified DEDUCE algorithm in Section 3.3.2), then CHOOSEQ calls itself recursively, invoking the corresponding lower level hypothesis space and setting this pattern as the leading hypothesis which it will then try and establish in this level. It is important to note that different frames of discernment are associated with different hypothesis spaces in the network. Also, the procedures for returning from a level (hypothesis space) to the one it was invoked from depends on the EXITCHK conditions as discussed in Section 3.3. The next section presents an example to illustrate the various features of the inferencing mechanism discussed above.

3.3.5. *An example*

This example, taken from the XX system[24], illustrates the functions of the five components of the inferencing mechanism. The system is currently trying to establish the site of a hydrocarbon play. Readers are referred to figs 2(a) and (b) that describe a section of the rules and the portion of the network that is relevant to the inferencing process being discussed. Initial evidence provided by the user results in the exact belief function:

$$m(\{h_1,h_2\}) = 0.45,\ m(h_1) = 0.25,\ m(h_2) = m(h_3) = 0.1$$

where $h_1 = (<$ site of play$>$ $<$margin $>)$, $h_2 = (<$ site of play$>$ $<$shelf $>)$, and $h_3 = (<$ site of play$>$ $<$craton $>)$. Based on these values, GETMAXH establishes h_1 as the leading hypothesis, and CHOOSEQ is invoked to pick an appropriate query to obtain more evidence in support of $<$ margin $>$ from the user. CHOOSEQ examines the rule network (fig. 2(b)) and picks ($<$ dist$>$ $<$less_equal_200 $>$) as the best evidence-node. Since this is an askable attribute, the system queries the user and determines the distance of the play from the margin is *less than 200 miles*. This causes rule 03 (fig. 2(a)) to fire and belief values get updated, according to eqn 4, as shown below:

m_{rul03} m	$\{h_1,h_2\}$ (0.8)	Θ(0.2))
$\{h_1,h_2\}$ (0.45)	$\{h_1,h_2\}$ (0.36)	$\{h_1,h_2\}$ (0.09)
h_1 (0.25)	h_1 (0.2)	h_1 (0.05)
h_2 (0.1)	h_2 (0.08)	h_2 (0.02)
h_3 (0.1)	Φ (0.08)	h_3 (0.02)
Θ (0.1)	$\{h_1,h_2\}$ (0.08)	Θ (0.08)

The updated belief function is:

$$m(\{h_1,h_2\}) = 0.576,\ m(h_1) = 0.272,\ m(h_2) = 0.109, \text{ and } m(h_3) = 0.022.$$

The GETMAXH function again identifies h_1 as the leading hypothesis. To further increase belief in h_1, CHOOSEQ determines that it will have to establish evidence corresponding to the *AND* node in fig. 2(b), which involves establishing two verifiable attributes. Let us assume that the system has established that there is *no abrupt change in slope*, and *as we move seaward the beds dip seaward.* It now descends to Level 2 to determine the *direction of deepening of the beds*. Again fig. 2(b) indicates a number of askable evidence patterns to establish that *beds deepen seaward*. CHOOSEQ first selects a query to determine the direction in which sediments become finer, and the user responds *seaward*. This establishes the hypothesis ($<$ beds_deepen$>$ $<$seaward $>$) with a belief value of 0.7. In a traditional system, additional rules would have been triggered automatically, and belief values of the $<$ site_of_play $>$ attribute would have been recomputed. However, in this case, establishing additional properties, such as *homogeneity of sediments* and *deepening of fauna* would have increased belief in the hypothesis ($<$ beds_deepen$>$ $<$seaward $>$), and would have resulted in different belief values for the $<$ site_of_play $>$ attribute.

In order to avoid repeated computation of belief values, which is expensive, the system does not propagate belief values out of a hypothesis space till the EXITCHK conditions are satisfied. In this example, the system continues to query the user till the belief value for ($<$ beds_deepen$>$ $<$seaward $>$) becomes 0.96, and the EXITCHK condition is established. This value is then propagated to the higher level, where belief values are updated in the *site of play* frame of discernment.

4. The system designer interface

This section summarizes mechanisms developed for MIDST to facilitate the task of knowledge base creation and modification. Functions for structuring the overall knowldege base as sequential partitions, entering and editing rules, and obtaining queries for askable evidence are discussed. It is assumed that the system designer, by prior interactions with the domain expert has designed the knowledge base partitions, evidence patterns and hypotheses are already represented as attribute-value pairs, and rules have been formulated in the format described in Section 3.1.

The system designer interface is implemented on APOLLO DN3000 and Sun 3 workstations, and runs under CommonLisp. This programming environment provides support for the display and use of multiple windows on the screen. The salient characteristics of the interface are: (i) a multiple window display, (ii) the use of special templates to facilitate rule entry, and (iii) simple mouse-controlled operations that allow the user to interact with the system in multiple windows simultaneously.

Creation of the domain knowledge base is performed in two distinct phases. In the first phase, the rule editor captures the overall problem-solving methodology of the expert, *i.e*, it makes the system designer specify sequential partitions for each subtask of the problem-solving process. The aim is to obtain sequential partitions (if any), and the hierarchical relationship between them. The skeletal structure of the knowledge base is displayed in the *structure window*. The process of defining the partitions and determining the hierarchical or sequential order is guided by a question-answering session with the system designer. After this phase is complete, the structure window is placed on one corner of the screen, so that the system designer may refer to it if he wants.

In the second phase, the system designer first specifies the goals *i.e.*, the list of final conclusions for each partition, enters rules that link evidence patterns to conclusions, and enters questions for evidence that is to be derived by directly querying the user. MIDST provides the system designer the facility to switch between the two phases at any time during the interaction. While entering rules for a partition, he may want to modify the sequential partition structure, and then return to the process of entering rules for the current partition. For the current partition in which rules are being entered, a *partition window* displays the partition id, the final conclusions for that partition, and the Lisp form of the last rule that was entered. A rule template is displayed in the *interaction window*, the window in which current interactions are occurring. Rules and the associated belief values are entered into the template. The user is permitted to enter a number in the interval [0,10] to express belief in conclusions. The basic purpose here is to obtain a relative ranking of the conclusions. These values are then converted to a belief function in the Dempster-Shafer framework. For every individual piece of evidence, the rule editor prompts the system designer for an associated query. This appears as a separate template within the interaction window. The list of queries entered for the

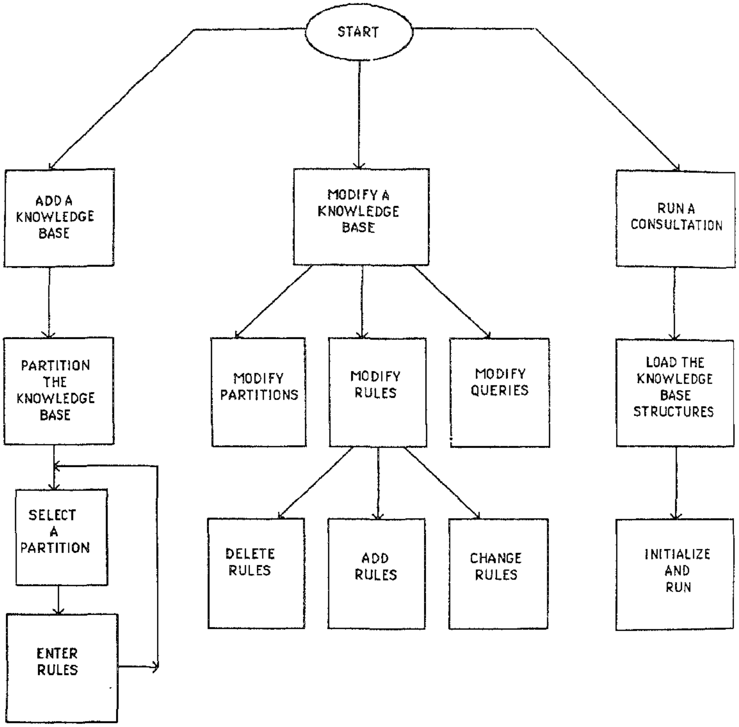

FIG. 5. Operations of the rule editor.

current partition stored in the *query window*. In addition, the system designer can open up the query window to independently edit the queries.

Once the system designer has entered all the rules and the associated queries, the rule editor separates all evidence patterns into askable and verifiable patterns. The verifiable evidence patterns do not have queries associated with them, and thereby need to be derived by firing rules. For each verifiable pattern there must be a set of rules whose RHS contains this pattern. If no rule is found, the system designer is informed and he either converts the evidence pattern to the askable type by providing a query, or he enters rules that can be used to derive the pattern.

The capabilities and sequence of operations of the current version of the rule editor are summarized in fig. 5. The rule editor allows the system designer to add a new knowledge base, modify an existing one or run a consultation to obtain debugging feedback. This facility to run a consultation is provided to the system designer to test the knowledge base by running sample case studies. MIDST allows the entry of large knowledge bases to be distributed over a number of sessions. The system designer can run test cases to check the validity of newly entered or modified knowledge, thus making it possible to do incremental debugging. The system designer can then go back to the expert for more directed consultations and discussions to refine the knowledge base.

4.1. *Adding a knowledge base*

The three major structures that the system designer needs to specify in the knowledge base are the sequential partitions, the rules within each partition and the queries associated with askable attributes. MIDST begins by explaining to the system designer the purposes and benefits of partitions, and then interacts with the system designer to obtain partitions and the relationships between partitions. As an example, consider the system designer entering the domain knowledge base for the OASES system[22]. The rule editor prompts are depicted in bold font and the system designer's input is in italics.

Would you like to enter a new knowledge base?
yes
Enter a name for your knowledge base.
OASES
Would you like to partition the knowledge base?
yes
How many steps have you divided your problem-solving process into?
3
How many partitions do you need at step 1?
1
Enter a name for the partition.
process type
How many partitions do you need at step 2?
5
Enter names for the 5 partitions.
Separate the names by commas.
continuous flow, batch flow, worker paced, machine paced, job shop
To which partitions in step 2 is the partition process type in step 1 connected?
all.

This information is displayed to the system designer in a separate window (the structure window). Window management is done from Common Lisp routines using underlying operating system calls. Currently the partition information is displayed in a non-graphic mode; however, this display will be improved in future versions. After relationships between partitions in steps 2 and 3 have been obtained, the rule editor enters the rule entry phase to allow the system designer to enter rules for each partition. The designer is required to input the LHS pattern of a rule, and then the RHS conclusions, and the rule editor converts the rule into the internal Lisp format. When the system designer supplies rules which negate a hypothesis, the appropriate conversion is done automatically. For example, consider the system designer entering an OASES rule:

Enter the attributes and values of the LHS pattern:
(separate attributes and values by a comma; one attribute-value pair per line)
machinery speed and size, not in good balance.

Enter the attributes and values of the RHS conclusions
(one conclusion per line; an attribute followed by a set of values separated by commas)
cause, capacity planning, process design
cause, workforce, maintenance.

In order to extract the expert's belief in the RHS conclusions, given the LHS evidence, the rule editor prompts the system designer to rank the RHS conclusions on a scale of 0–10. Note here that the expert is not supplying an absolute support or belief value for the conclusion, but is merely providing a relative ranking based on his judgement.

Enter the relative ranking for the conclusion on a scale 0–10.
9 3

A very desirable feature of the D-S framework for representing knowledge in uncertain domains is the explicit definition and representation of ignorance. In formulating the rule, the editor specifically queries the expert (system designer) about his belief in the relevance of the LHS evidence, *i.e.*, given that the system designer will be examining other evidence for making conclusions in this frame of discernment, on a scale of 1–10 to what extent does this contribute to making a final conclusion.

On a scale of 1–10 what is the relevance of this evidence in the overall reasoning process?
8

From this, the system computes $m\ (\Theta)$ for the belief function corresponding to this rule to be 0.2 (*i.e.*, 1–8/10). The relative ranking supplied by the expert is then normalized to yield the belief values (the m function) for the rule according to the equation:

$$(bf)_{new} = (bf)_{old} \times \frac{1 - m(\Theta)}{\sum (bf)_{old}} . \tag{5}$$

Using $m\ (\Theta) = 0.2$, rule editor formulates the following rule:

[(< machinery_speed_and_size > < not_in_good_balance >) BF] →
{[(<cause> <capacity_planning> <cause> <process_design>) 0.6]
[(<cause> <workforce> <cause> <maintenance>) 0.2]}.

The m (Θ) value along with the above values represents the measure of exact belief function for this rule.

The rule in its Lisp form along with the associated belief values is displayed in a separate window. For each new attribute entered in the LHS of a rule, the rule editor queries the system designer for an associated query unless one has already been entered. This query is displayed in a second window. Thus, while the system designer is entering rules, two separate windows (besides the interaction window) are open displaying all the rules that have been entered and all the associated queries. When the system designer indicates that he has finished entering rules within a partition the rule editor asks him to specify the ASKFIRST queries, if any, for that partition. Finally, rules specifying the transfer of control from the current partition to the partitions it is linked at the next step are obtained from the system designer.

4.2. *Modifying the knowledge base*

As fig. 5 indicates, facilities are provided in MIDST to modify the overall structure of the knowledge base, the rules within partitions, and the associated queries. Modifying the overall structure involves the addition or deletion of partitions, and the addition, deletion or modification of links between partitions. While deleting and adding partitions, or deleting and adding links care needs to be exercised that the hierarchical relationship of the existing partitions is not violated. When a new partition is added,the rule editor automatically prompts the system designer to enter the rules within that partition.The modified structure is again displayed to the system designer in the structure window.

Modifying rules similarly involves the addition of new rules, the deletion of rules and the modification of existing rules. In the current prototype no sophisticated editing facilities are available. Therefore, in order to modify a rule the system designer is required to enter the complete rule all over again. Facilities for changing part of a rule, or changing only the belief values have not yet been implemented. Whenever the rule set is modified, MIDST automatically recompiles the network and stores it for future use. The system designer can also modify queries (*i.e.*, change the language of a query), add new queries (this may make a hitherto verifiable attribute into an askable attribute), or delete an already existing query. The addition and deletion of queries also requires that the rule network be recompiled. Usually this recompilation is done, just before the system designer wants to run a consultation to see the effects of the modifications.

4.3. *Running a consultation*

This involves three steps as illustrated in fig. 5. The first step involves loading all the knowledge base structures such as the rules, the queries and the rule network. In the second step, the working memory structures and the book-keeping fields in the network are appropriately initialized. The third step is the inferencing procedure described in Section 3.3.

5. Conclusions

In this paper, we have described the design and implementation of MIDST, an expert system shell for mixed initiative reasoning. A primary achievement of our research has been the implementation of an efficient technique for incorporating the Dempster-Shafer evidence combination scheme for inexact reasoning. This was accomplished by compiling the domain rules into a rule network. We consider the design and implementation of the network to be the major contribution of this research effort. The network was used not only to speed up the backward-chaining process, but also to partition the overall knowledge base into hypotheses spaces similar to a method suggested by Gordon and Shortliffe[20]. In addition to computational advantages of a smaller frame of discernment, the network allows implementation of the D-S scheme without compromising the basic assumptions of independent pieces of evidence and a mutually exclusive hypotheses set.The network provided an efficient and convenient means of propagating belief values between levels. The compilation of the network is done off line and the resultant network is stored in a file. The network construction algorithm was designed in a manner that it runs in $O(n)$ time; where n is the number of rules in a partition[27]. The rules and the network for individual partitions are stored in separate files, and loaded only when a partition becomes active. This speeds up the loading of large knowledge bases.

A second task addressed was the design and implementation of knowledge base construction tools. Here, special attention was paid to effectively utilize features of the underlying programming environment. The internal Lisp representation is made opaque to the system designer, and the acquisition of belief values is based on a framework that we hope experts will easily relate to. MIDST currently runs on Sun 3 and Apollo DN3000 workstation under Common Lisp. Applications of MIDST currently underway include porting OASES[23] and developing the XX system[24].

There are a number of tasks that we envision will improve the current version of MIDST. They are:

1. Extensions to the rule editor to incorporate sophisticated editing facilities, provide graphic displays and the ability to switch from the rule entry mode to a mode where the system designer can run a consultation, modify rules and then switch back to the rule entry mode. In addition, a TEIRESIAS[14] type checking of the rules is also envisaged.
2. The development of sophisticated explanation mechanisms and debugging aids.
3. Extending the network to facilitate its use during forward chaining. This can be done by specifying additional dummy nodes.
4. The propagation of incremental belief values in the Dempster-Shafer framework.
5. Extending GETMAXH so that it ranks hypotheses not on the basis of the absolute belief but on the basis of the plausibility interval. This will necessitate the generalization of computational techniques discussed by Shenoy and Shafer[26].
6. Develop tools that will aid the system designer in building specialized user interfaces.

References

1. HARMON, P. AND KING, D. *Expert systems: Artificial intelligence in business*, John Wiley, New York, 1985.

2. BUCHANAN, B.G. AND SHORTLIFFE, E.H. (EDS) *Rule based expert systems: The MYCIN experiments of the Stanford HPP*, Addison-Wesley, Reading, MA, 1984.

3. BENNETT, J.S. AND HOLLANDER, C.R. DART: An expert system for computer fault diagnosis, *Proc. 7th. Int. Jt. Conf. Artif. Intell.*, 1981, Vol. 7, pp. 843–845.

4. MCDERMOTT, J. R1: An expert in the computer systems domain, *Proc. Am. Ass. Artif. Intell.*, 1980, Vol. 1, pp. 269–271.

5. DUDA, R.O., GASCHNIG, J. AND HART, P.E. Model design in the PROSPECTOR consultant system for mineral exploration, in *Expert systems in the microelectronic age*, Edinburgh Univ. Press, 1979, pp. 153–167.

6. LINDSAY, R., BUCHANAN, B.G., FEIGENBAUM, E.A. AND LEDERBURG, J. *DENDRAL*, McGraw-Hill, New York, 1980.

7. BENNETT, J.S., CREARY, L., ENGELMORE, R. AND MELOSH, R. *SACON: A knowledge-based consultant for structural analysis*, Report no. HPP-78-23, Computer Science Department, Stanford Univ., CA, 1978.

8. VAN MELLE, W. A domain-independent production rule system for consultation programs, *Proc. 6th Int. Jt. Conf. Artif. Intell.*, 1979, pp. 923–925.

9. WATERMAN, D.A. AND HAYES-ROTH, F. An investigation of tools for building expert systems, in *Building expert systems*, F. Hayes-Roth, D.A. Waterman, and D.B. Lenat(eds), 1983, pp. 169–215, Addison-Wesley, Reading, MA.

10. WEISS, S.M. AND KULIKOWSKI, C.A. *A practical guide to designing expert systems*, Rowman and Allenheld, Totowa, NJ, 1984.

11. FAIN, J. *et al* *The Rosie reference manual*, N–1647–ARPA.RAND Report, Santa Monica, CA, 1981.

12. FORGY, C. AND MCDERMOTT, J. OPS: A domain-independent production system language, *Proc. 5th Int. Jt. Conf. Artif. Intell.*, 1977, pp. 933–939.

13. ERMAN, L.D., LONDON, P.E. AND FICKAS, S.F. The design and an example use of Hearsay III, *Proc. 7th Int. Jt. Conf. Artif. Intell.*, 1981, pp. 409–415.

14. DAVIS, R. *Use of meta level knowledge in the construction and maintenance of large knowledge bases*, Ph.D. Thesis, Dept. of Computer Science, Stanford University, 1976.

15. SZOLOVITS, P. AND PAUKER, S.G. Research on a medical consulting system for taking the present illness, *Proc. Third Illinois Conf. Medical inf. Systems*, Univ. of Illinois at Chicago Circle, 1976, pp. 299–320.

16. POPLE, H. The formation of composite hypotheses in diagnostic problem solving—an exercise in synthetic reasoning, *Proc. Int. Jt Conf. Artif. Intell.*, 1977, Vol. 5, pp. 1030–1037.

17. Szolovits, P. and Pauker, S.G. — Categorical and probabilistic reasoning in medical diagnosis, *Artif. Intell.*, 1978, **11**, 115–144.

18. Shafer, G. — *A mathematical theory of evidence*, Princeton Univ. Press, NJ, 1976.

19. Adams, J.B. — Probabilistic reasoning and certainty factors, *Math. Biosci.*, 1976, **32**, 177–186.

20. Gordon, J. and Shortliffe, E.H. — A method for managing evidential reasoning in a hierarchical hypothesis space, *Artif. Intel.*, 1985, **26**, 323–357.

21. Siler, W., Buckley, J. and Tucker, D. — Nonmonotonic fuzzy logic: Experience with FLOPS, *Proc. NAFIPS-86*, Bandler, W. and Kandel, A. (eds), New Orleans, LA, 1986, pp. 524–530.

22. Biswas, G., Abramczyk, R. and Oliff, M.D. — OASES: An expert system for operations analysis – The system for general cause analysis, *IEEE Trans.*, 1987, **SMC-17**, 133–145.

23. Biswas, G., Oliff, M. and Abramczyk, R. — OASES: An application in fiberglass manufacturing, to appear in *Int. J Expert Systems*, 1988, **3**,

24. Anand, T., Biswas, G., Pai, M., Kendall, C., Cannon, R., Bezdek, J. and Morgan, P. — XX: Hydrocarbon exploration using a knowledge-based approach, *Proc. AAPG Annual Conv.*, 1988, Houston, TX, March 20–23.

25. Barnett, J.A. — Computational methods for a mathematical theory of evidence, *Proc. 7th Int Jt Conf. AI*, 1981, pp. 868–875.

26. Shenoy, P.P. and Shafer, G. — Propagating belief functions with local computations, *IEEE Expert*, 1986, **1**, 43–52.

27. Anand, T.S. — *MIDST: An expert system shell for mixed initiative reasoning*, M.S. Thesis, Dept. of Computer Science, Univ. of South Carolina, 1987.