PROCEEDINGS OF

THE WORLD CONGRESS ON EXPERT SYSTEMS℠

Orlando, Florida
December 16-19, 1991

Jay Liebowitz

Editor

Volume 1

PERGAMON PRESS
New York • Oxford • Seoul • Tokyo

Multiple-Expert Systems

Umesh Mittal and James Geller
Department of Computer and Information Science
New Jersey Institute of Technology
Newark, N.J. 07102, USA
geller@mars.njit.edu

ABSTRACT

One characteristic of many current expert systems is the expectation of consistency of their knowledge bases. The lack of this consistency could cause unpredictable results, but consistency is undesirable because it leads to inflexible reasoning. Research in the area of human decision making suggests that a human decision is not always made based on the advice of a single expert. Therefore, it makes sense to design a system where the knowledge of all the experts is available at the decision making stage, even if it is inconsistent. In this research, we present the issues involved in the development of a Multiple-Expert System shell. Such a shell and a testbed application have been developed and will be described here.

INTRODUCTION

During the process of building an expert system a knowledge engineer often acquires knowledge from multiple sources. Unfortunately, there are often issues on which experts hold different opinions resulting in conflicting expertise. The knowledge engineer ensures the elimination of conflict and unwanted interactions or inconsistencies in the resulting knowledge base which is then encoded in the system [2]. Since decisions are seldom based on the advice of a single expert, a system in which the knowledge of more than one expert is available on-line at all times would provide a more exact model of real life reasoning. In this paper we present one such system called Multiple-Expert System along with a prototype application that shows its usefulness and robustness.

RELEVANT WORK

The problem of coding expert systems based on the knowledge of more than one expert has been dealt with in many publications. Wolf [10] emphasizes the need to use more than one expert's knowledge to construct a meaningful system. Shaw *et al.* [8] in their paper on validation mention that in many areas, each expert has a unique perspective on the topic and thinks that there is only one way that the topic should be considered. During knowledge acquisition for IPM [10] fundamental differences were discovered among experts. The knowledge acquisition process should utilize many knowledgeable sources of expertise as opposed to dealing with a single "*world-class*" expert [2]. One of the integration issue in knowledge acquisition systems is integrating represented knowledge of various experts into a model which shows the *conflicting* views of the domain and integrating diverse knowledge sources in their respective representations [4].

Conflicts and inconsistencies become important when we have multiple sources of knowledge. Shaw *et. al.* [8] discuss the disagreement among multiple sources of knowledge. These sources often have different perspectives regarding solutions to a specific problem. The existence of conflict, however, is not an undesirable situation [2, 8]. The decision maker has to deal with this conflict in advice and make a decision. The process by which this decision is made is ill-defined and subjective [1, 6].

There are *organized* group decision making situations in which individuals argue different sides of an issue. On the other hand, there are *ad hoc* situations where a decision maker encourages group discussion to consider various viewpoints and/or alternatives [2]. The two categories of techniques most often applied to group decision making situations are group processes and mathematical aggregation. Examples of group processes include the Delphi Method [3] which is an iteration of questionnaires and Normative Group Technique and Social Judgement Theory which is an estimate-talk-combine-rank process. Statistical averaging/weighting and Bayes' Rule are examples of mathematical aggregation techniques [2]. One of the inherent strengths of group decision making processes is their ability to adapt to slight differences among the decision making situations and select the solution which suits the problem best.

Various methods have been described in the literature to resolve the conflicts between experts. The approach used in [10] is to discuss the differing points of view and finally arrive at a single "best" solution. In case of failure of this method the coordinator's word is final. Many expert systems use various quantitative measures of evidential support to select one opinion over another. Gaglio *et al.* [1] assign a fuzzy value to each expert opinion whereas Reboh requires the assignment of weights to experts [6]. But then the user has no understanding of how these numbers are arrived at [7].

The research in the field of conflict resolution has been broadly divided into cooperative and competitive groups [2]. The classical approach to the problem of conflicting expertise is representative of cooperative group decision making processes where consensus is often achieved as a result of compromise

or conformance of individual expertise. An individual opinion is likely to be a good solution if it withstands the scrutiny of a group. It may, however, not be an appropriate solution for similar but slightly different problems [2].

MULTIPLE-EXPERT SYSTEM

By seeking consensus before building a knowledge base the contemporary expert system becomes less finely "tunable" to the problem. Replacing the individual experiential reasoning with the legislated collective reasoning results in the inability of the system to reliably reproduce the result of cooperative group decision making processes [2]. Also a judgement is required on the part of the knowledge engineer in the contemporary expert system to force consensus even though he may have little expertise in the specific domain. LeClair [2] emphasizes the importance of storing knowledge from all the experts in the system without any effort to resolve any occurring conflict.

In the Multiple-Expert System presented in this paper, there is no need for a knowledge engineer to pass any kind of qualitative judgement on the expert's knowledge. This system is capable of storing knowledge of all the experts without any modification. The knowledge contents is not looked upon until run-time. A Multiple-Expert System shell has been developed at the CIS Department of NJIT using C-Prolog on a SUN workstation. This section is a discussion on the various problems which are unique to the Multiple-Expert System and were discovered and solved during this work.

Knowledge Representation

Our knowledge representation is based on first-order logic which is supported by Prolog. We have devised a process of **belief encapsulation** to overcome the problem caused by the inability of all logic based knowledge representation techniques to store conflicting pieces of knowledge. This process needs to be performed on each item of knowledge before it is made part of the knowledge base. It successfully hides all possible contradictions among the knowledge bases of different experts from the underlying logic system without making any modification in any knowledge base. We assume that there is no conflict within the knowledge base of any **one** expert.

The process of encoding the knowledge in Prolog consists of locating keywords in a natural language sentence and forming valid Prolog clauses from them. A keyword can be roughly equated to a content word, as opposed to a function word. One of these keywords is identified as predicate and the others become arguments. Thus a natural language sentence is converted into a Prolog atom. We do not provide any Natural Language features in our system.

Each of these Prolog atoms is now belief encapsulated before storing it in the knowledge base of the Multiple-Expert System shell. In belief encapsulation, each piece of knowledge is made part of a new atom which is made up of two arguments [5]. This atom has the word **belief** as the predicate and the two arguments are the name of the expert and the Prolog item of knowledge generated by the expert. The name of the expert is also needed to display the answers and possibly for conflict resolution between experts.

Example

The following examples show the *belief encapsulation* of clauses and rules. An expert clause `expert_clause` would be encapsulated as:

```
believes ( expert_name, expert_clause)
```

However, if the knowledge item is a rule then all its components need to be individually encapsulated. The rule:

```
consequent(Argument_1,Argument_2) :-
        antecedent_1(Argument_1),
        antecedent_2(Argument_2).
```

would be encapsulated as:

```
        believes( expert_name,
consequent(Argument_1, Argument_2)) :-
```

```
believes( expert_name, antecedent_1(
            Argument_1)),
believes( expert_name, antecedent_2(
            Argument_2)).
```

Implementation

Knowledge encapsulation is the first task performed by the shell. The only requirement on the knowledge from an expert is that it should be a syntactically correct Prolog program with some additional restrictions on the syntax used. However, since the whole knowledge content is encapsulated, the Prolog primitives require additional processing on the part of the shell for proper execution. Therefore, not all the Prolog primitives are supported in the current version. Of the side-effecting primitives only `write`, `read`, and `nl` are currently supported. The program can be easily modified to accommodate many other primitives.

Minimizing Search Space

The knowledge base of a Multiple-Expert System is expected to be very large because we are keeping the knowledge of several experts available at all times. If there is a piece of knowledge that all experts agree upon, it would still be stored independently for each expert. Since it is not likely that, in a big domain, every expert would have an opinion on everything, there will be some items of knowledge which only one expert maintains. In a situation like this, restricting the search space to relevant experts is very necessary. Any time spent in searching the knowledge base of an expert who does not have an opinion on the query is a waste.

Evaluation of a minimum number of experts is the technique used to minimize the time required to obtain an answer. It avoids search in the knowledge bases of the experts who do not have an opinion on a given question. During the construction of the knowledge base all the keywords used by an expert are identified and his name is attached to them. When a query is presented to the Multiple-Expert System shell it identifies the keywords in it and finds the names of all the experts who have used all the keywords in the query. These are the only experts who are likely to have an opinion on the query and we search the knowledge-base only for these experts.

Implementation

A keyword file is created while reading the knowledge bases of the experts. Every instance of a keyword is stored in this file along with the name of the expert using it. The Multiple-Expert System shell then reads this file and prepares a list of experts for each unique keyword who are using the keyword. The format of this list is:

keyword ([expert1, expert2, . . . , expertn])

This list is then asserted into the system. The list of experts using a keyword can now be found by a Prolog query. An intersection of the lists for all the keywords gives the list of experts who have used all the keywords in the query at least once. A query to the interpreter can now be qualified by the names from this list to reduce the search space in the knowledge-base.

Keyword Storage and Retrieval

A keyword search is performed every time a query is passed to the interpreter. Therefore it is very important to make this operation fast. The best results we could achieve using C-Prolog was by asserting clauses in the interpreter and letting it do the search. It was found by experiment that using linear lists is a less efficient method. Trees require comparison operators for storage and retrieval of information. C-Prolog provides comparison operators only for numbers and not for keywords which are made up of letters. This makes use of tree implementations less efficient. Hashing techniques require constant time data structures which are not supported by C-Prolog. Thus the method of storage and retrieval used by us is the best choice for C-Prolog.

Use of Thesaurus

In natural language different words can be used to refer to the same concept. Also a word often has different meanings depending upon the context. Since the only limitation on

the contributing expert's knowledge is syntactic correctness in a well-defined subset of Prolog, it is possible to find different words being used by experts to refer to the same concepts. Without the use of any special technique a user would get the opinion of only the experts with whom he shares the vocabulary of that query because a matching algorithm would fail to match two words that are not identical.

We want a complete answer which maintains the fine distinction in the original words. The problem of matching two different atoms, if they have the same meaning in the given context, is unsolvable at the current state of the art. We have devised a partial solution using a thesaurus. The system compares a word in the query with all its possible synonyms in the knowledge base. The thesaurus, a Prolog program with each clause defining the similarity relation between the two words using *thesaurus* as the predicate, is stored in a file. The word used as predicate, however, cannot be tested for synonyms in this scheme of operations.

Example

If two experts enjoy the game of baseball and say:

Expert 1: `It is fun watching the game of baseball.`
Expert 2: `I enjoy watching the game of baseball.`

This can be encoded in Prolog as:

Expert 1: `baseball(fun).`
Expert 1: `baseball(enjoy).`

The user may use another word for the same concept. Consider the example:

User Query: `How many experts like the game of baseball?`
Prolog Code: `baseball(like).`
Expected Reply: `The expert1 agrees with the statement baseball(fun)`
`The expert2 agrees with the statement baseball(enjoy)`

This is because `fun`, `enjoy` and `like` convey similar thoughts. Asserting the two opinions or Prolog facts and presenting the third as a question would result in a wrong answer implying that nobody likes the game of baseball. This can be avoided by using the thesaurus.

Implementation

After a keyword is selected to be checked for synonyms, a new query is formulated with the keyword being replaced by a variable. The keyword and its position need to be memorized for future processing. This new query is passed to Prolog which generates all possible answers. The resulting list of answers is searched to obtain the expert opinion using a synonym by testing each instantiation of the variable for the *thesaurus* relation.

Predicate Transparency

This problem arises at the stage of converting natural language sentences to Prolog terms before they are encapsulated. There is no natural way to select a keyword as predicate and it is possible that different experts would select different keywords as predicates even if the set of keywords identified in the natural language sentence is identical. As a result only a part of the knowledge stored in the system may be accessible to the user.

Predicate transparency is implemented to solve the problem of selection of a predicate from a set of keywords. The user should not be forced to guess the keyword that an expert has selected as the predicate. It is the responsibility of the shell to try other keywords as predicates and display all the valid expert opinions so obtained. All the keywords in the user query need to be tried as predicates to extract all possible answers from the knowledge base.

Example

For example, if three experts agree on the fact that baseball is a sport and that it is fun playing it then this can be expressed in natural language as:

Expert1: `Baseball is a sport and it is fun playing this game.`
Expert2: `It is fun playing baseball which is a sport.`
Expert3: `Sports is fun, especially when we are playing baseball.`

These three sentences could be coded as:

Expert1: `baseball(sport,fun).`
Expert2: `fun(baseball,sport).`
Expert3: `sport(fun,baseball).`

A user query can now be represented as:

User Query: `sport(fun,X).`

Without predicate transparency an incomplete conclusion will be displayed indicating that only the third expert agrees.

Implementation

Our Multiple-Expert System shell provides interactive guidance by the user at query time to reduce the number of permutations involved in a complete algorithmic implementation of predicate transparency. The user is presented with the old predicate and then the system prompts the user with the remaining keywords in the original query to select one which should be tried as new predicate. A new query is created by exchanging the original keyword and the selected keyword.

To do this, the original query is broken into a list of keywords, the list is modified to create a new list with the original predicate and the selected keyword interchanged. The new list can now be converted back into a new query. This new query is then passed to the Prolog interpreter for generating all possible answers to this query. Each answer to this query is assumed to be an answer to the original query as well and is therefore displayed to the user.

Generating Response: A Summary

The Multiple-Expert System shell when executed on top of the Prolog interpreter first reads in the three system files *viz.* the encapsulated knowledge base, the thesaurus and the keyword database. The keyword database is then converted into one list which is asserted in the internal Prolog database. The user is now prompted to enter a query.

Then the shell finds the names of all the experts who are using all the keywords in the query and puts them into a list. For each name in this list, the shell forms a goal for the interpreter by belief encapsulating the user query with this name. This goal is now passed to the Prolog interpreter and all possible instantiations of any variable contained in it are found. The information obtained is used in the last step to display the answers in a proper format.

After the answers to the original query are displayed the user has a choice to try the thesaurus operations or to use the same query with a different predicate. In either case he is presented with all the keywords of the original query to select one and the appropriate operations are performed as explained in the previous sections.

PASCAL DEBUGGER

An application of the Multiple-Expert System shell has been developed to demonstrate its usefulness and robustness. In this application we have created the knowledge base for three experienced programmers whose expertise lies in debugging the errors occurring while developing a Pascal program using a commercially available Turbo Pascal compiler. It simulates the capabilities of all the three experts and provides the user with good directions for solving programming errors. With this system the user would enter the error number and the type of error and he would be provided with the opinions of all the three experts.

One of the advantages of having the debugging knowledge of more than one expert for a beginner's program is that even if there is only one solution for a problem, it can be presented to the user in many different ways. A multitude of approaches towards solving a programming problem is expected to "better" help a learner than one legislated advice. The user can select any opinion depending upon his own criterion. He could select the one which is being espoused by the majority of experts or the one suggested by an expert whom he found most helpful in his previous interactions with the system.

Turbo Pascal Error Messages

There are two types of errors generated by Turbo Pascal [9], *Compiler Errors* and *Run-time errors*. The count of errors for each of these two types is 145 and 38 respectively. For example:

Compiler Error: `Error 15: File not found (WINDOW.TPU).`

Run-time Error: `Run-time error nnn at xxxx:yyyy.`

where *nnn* is the run-time error number, and *xxxx:yyyy* is the physical memory address where the error occurred in the format *segment:offset*.

Compiler errors can be sub-divided into categories. These are syntax errors, errors due to compiler limitations, errors caused by the underlying system (DOS), environment related errors and language limitations. Different types of run-time errors are DOS errors, I/O errors, critical errors and fatal errors. It is not always possible for a beginner to infer the meaning a short error message.

Use of Multiple-Expert System Shell

This section explains the appropriateness of the domain as an application of the Multiple-Expert System shell. We show that all the problems identified *viz.* existence of differing expert opinions, need for thesaurus and predicate transparency, do exist in this domain and can be solved by the solutions explained in the previous section.

Differences in Experts' Opinions

Turbo Pascal defines a large number of errors. Some of the errors have very simple solutions according to all three experts. On the other extreme there are some errors for which every expert has his own point of view. In between these two extremes there are error conditions for which only two experts agree on one approach. The following four tables demonstrates all these possibilities using the expert opinions from the knowledge base used in the application.

Table - 1
Completely Independent Opinions

Compiler Error 1	Out of memory
Expert 1	If Compiler/Destination or Option/linker/link options are set to memory, set it to disk in the integrated environment and compile again.
Expert 2	Use TPC.EXE program which takes less memory or break program into smaller units.
Expert 3	Remove all the memory resident programs that you might be using and try again.

Table - 2
Completely Agreeing Opinions

Run-time Error 159	Printer out of paper
Expert 1	The system is trying to print. Put paper in the printer
Expert 2	The system is trying to print. Put paper in the printer
Expert 3	The system is trying to print. Put paper in the printer

Table - 3
Partially Agreeing Opinions

Compiler Error 61	Invalid typecast
Expert 1	The sizes of variable reference and destination type is different in a variable typecast, which is not allowed.
Expert 2	You are not allowed to typecast an expression where only a variable reference is allowed.
Expert 3	The sizes of variable reference and destination type is different in a variable typecast, which is not allowed.

Table - 4
Similar but Different Opinions

Compiler Error 12	Type identifier expected
Expert 1	The identifier does not denote a type.
Expert 2	The system does not recognize this identifier as a valid type.
Expert 3	For type declaration use either a system defined or user defined type.

When the Multiple-Expert System sees the type of error and the error number it knows what type of error this is. For each category of errors, the system displays a general message and the opinion specific that error is then displayed. It is possible that different experts use different words for the same category of errors. An expert may categorize an error using the word `syntax` while the other might call it an error in `grammar`. Both these words need to be stored in the thesaurus. The system can can display the opinions of all the experts when the user enters an appropriate query. The thesaurus statement and a typical user query are shown below:

Thesaurus statement: `thesaurus( syntax, grammar).`

User Query: `error(syntax).`

The implementation of predicate transparency is also demonstrated using the error categories. An expert might use `error` as predicate to encode an error being caused by the limitations of Turbo Pascal. Another expert can select the word `limitation` to encode the similar information. A user should be able to find these two expert opinions without worrying about predicate selection. Our shell, because of the implementation of predicate transparency, can display opinions of all the experts. The expert opinions and the user query are shown below.

Expert1: `error( limitation, turbo).`
Expert2: `limitation( error, turbo).`
User Query: `error( limitation, turbo).`

CONCLUSION

We have implemented a prototype Multiple-Expert System shell using Prolog. It stores the knowledge of several disagreeing experts simultaneously and provides solutions for many of the above mentioned problems. A minimum number of sub-knowledge bases are evaluated for any query. The system implements a thesaurus and, therefore, finds answers even if the words used by the user and the experts are different but similar. It finds all the correct answers even if different predicates are used to represent the same piece of knowledge.

References

[1] Gaglio, S., Minciardi R., & Puliafito P. P. *Multiperson Decision Aspects in the Construction of Expert Systems*, IEEE Transactions on Systems, Man and Cybernetics, **SMC-15**, 4, 536-539, 1985.

[2] LeClair, Steven R. *Interactive Learning: A Multiexpert Paradigm for Acquiring New Knowledge*, SIGART Newsletter, 34-44, 1989.

[3] Linstone, Harold & Turoff, Murray *The Delphi Method: Techniques and Applications*, Addison Wesley, Reading, MA, 1975.

[4] Morik, Katharina *Integration Issues in Knowledge Acquisition Systems*, SIGART Newsletter, 124-131, 1989.

[5] Most J. Multiple Expert System, Unpublished report, Department of Computer and Information Science, New Jersey Institute of Technology, Newark, NJ, 1989.

[6] Reboh, R. *Extracting Useful Advice from Conflicting Expertise*, Proceedings of the Eighth International Joint Conference on Artificial Intelligence, William Kaufmann Inc., Los Altos, CA, 145-150, 1983.

[7] Sage, A. P., & Botta, R. F. *On Human Information Processing and Its Enhancement Using Knowledge-based systems*, Large Scale Systems, **5**, 35-50, 1983.

[8] Shaw, Mildred L. G., & Woodward, J. Brian *Validation in a Knowledge Support System: Construing and Consistency with Multiple Experts*, International Journal of Man-Machine Studies, **29**, 3, 329-350, 1988.

[9] Turbo Pascal, Reference Guide, Version 5.0, Borland International Inc., Scotts Valley, CA, 1989.

[10] Wolf, Walter A. *Knowledge Acquisition From Multiple Experts*, SIGART Newsletter, 138-140, 1989.