DETECTING AMBIGUITY: AN EXAMPLE IN KNOWLEDGE EVALUATION

D.W. Loveland and M. Valtorta

Computer Science Department
Duke University
Durham, North Carolina 27706. USA

ABSTRACT

Expert systems have been developed around one expert partly because the expert has been totally responsible for the soundness of the knowledge base. Without strong aids to help ensure soundness in building expert systems, we must rely on the soundness of the mature cohesiveness of a human expert. As knowledge bases grow this mature expertise will not be adequate. We propose a new method (for expert systems) to aid the expert in knowledge evaluation within the rule-based system setting. We consider the problem of ambiguity within a classification system as an example of the proposed technique.

Introduction.

Expert systems have depended upon the knowledge base of the collaborating expert for soundness and richness. However, as knowledge bases grow and try to remain current in a changing world, incremental knowledge needs to be added, and one does not have the filter of usage over time that makes an expert's knowledge so generally reliable. One is led to considering means of aiding the expert as he evaluates new information and attempts to update the knowledge base with this information. Many different types of aids will be useful; we suggest a useful approach and give a specific example within this approach.

There has been some realization of the importance of aiding knowledge acquisition. TEIRES1AS (see [Davis, 1982]) not only provided convenience tools for rule input but helped with rule integrity by use of "rule models." Michalski and Chilauski [Michalski and Chilauski, 1980] investigated inductive learning as a means of automating knowledge acquisition. Recently. Suwa, Scott, and Shortliffe [Suwa, Scott, and Shortliffe, 1982] have directly addressed the problems of verifying "completeness and consistency" in the context of the 0NCOCIN system. Also see this last paper for a good iummary of the need for and present status of knowledge acquisition aids.

Our general concern is also with the quality of the knowledge in a knowledge base. Our immediate ooncern is to aid the expert with feedback regarding unintended interactions with other rules in a rule-based system when a new rule is added. In this paper we assume reasoning with certainty. Extending the approach considered here to weighted evidence reasoning is an obvious next step.

This work Is supported in part by the Air Force Office of Scientific Research under grant AFOSR-81-0221.

The approach we suggest is supplemental to those previously studied. We suggest discovering properties for significant classes of problems and then defining procedures for quickly detecting violations of the properties. This puts semantic information into the evaluation process directly. As was learned in automated theorem proving, if the problem class you model is broad or important enough, and your device sufficiently effort-saving (here over standard testing), then the device will be useful. In the example we give, the property is important to many expert systems existing today and the procedure we suggest saves much effort.

To illustrate the modeling of a property we choose the ambiguity property within classification systems. Classification, or diagnostic, systems map certain inputs into their appropriate classes, such as certain symptom sets into the appropriate disease. Let us define an *atomic meaningful input vector* (atomic vector) as a set of attribute-value pairs that elicits a single output, called a *class*. Intuitively, these are usually the most important input vectors. If no inhibitor rules exist, then two or more atomic meaningful input vectors can combine to form a (general) *meaningful input vector* with output the union of the classes defined by the constituent atomic vectors. This superposition property is common: if one has ail the symptoms of a cold and a broken leg, one probably *has* a cold and a broken leg. We assume the property of superposition holds. (This can be softened.)

When the expert adds a rule to a rule-based system, he would like to know what changes he has caused. However, in a large system, testing every possible input vector may be impossible. He might well settle for knowing what attribute-value pairs are elements of vectors of each specific class, and if any previously atomic meaningful input vector now is a member of two or more classes. The latter problem we call the *ambiguity* problem," we give an efficient means of testing which returns attribute-value pair associations and helps the expert with the ambiguity problem.

We note that we have selected a specific property, or problem, to check within systems where exhaustive testing is too time-consuming. For large domains, there is a tremendous number of possible attribute-value combinations, most of them meaningless. We focus on analyzing the meaningful vectors, here taken to be those that belong to at least one (output) class. We have chosen one (important) problem over that class to illustrate that effective testing procedures can exist.

Along with natural devices we will exploit the following observation. By superposition, there will be

many ambiguous meaningful input vectors (any union of meaningful vectors of different classes). If we could obtain the minimally ambiguous *(min- ambiguous)* vectors, we need only display these to the expert; a now-ambiguous ex-atomic meaningful vector will be caught.

We model the inference system here by an "and-or" graph. (See Figure 1.) An Input node is a single attribute-value pair. An attribute may have several values. To represent this we partition the nodes into *sections,* each section denoting one attribute. Therefore, an input vector can list only one node per section. A *pseudovector* is any collection of input nodes, perhaps several per attribute. We assume that we can process pseudovectors; the classification of a pseudovector is all classes inferred by activating all nodes of the pseudovector. A rule is depicted by a dark node with conjoined input (cut by a curved arc) and one or more outputs. Output nodes are labelled CI, etc.

For simplicity, we limit ourselves to two classes. A *Ci-vector* is an input vector of class Ci, for i=1,2. A *12-node* (read "one-two node") is an input node in both a C1-vector and a C2-vector. V_{12} is the set of all 12-nodes. We eliminate most superposition ambiguous vectors by demanding that the ambiguous vector be composed of V_{12} nodes. (This is overly restrictive; we deal with some of this later, and must forego other possibilities here.) Figure 2 shows a min-ambiguous vector from V_{12} that is (probably) faulty. Figure 3 shows that min-ambiguous vectors from V_{12} need not be faulty. Vector <a b c> is a C1-vector and a C2-vector, which is probably intended. We simply have <a b>∈C1, <c d>∈C1, <a c>∈C2 and <b d>∈C2.

Finding unintended ambiguities.

Our concern is to locate unintended classifications for meaningful input vectors. Besides possibly using unusual input "vectors" (pseudovectors) for input, we need only to augment our inference system to execute "backflooding" in order to use the procedure discussed here. By *backflood from* Ci we mean to follow all backchaining paths and mark every Ci-vector input node by "i". This is amenable to parallel computation.

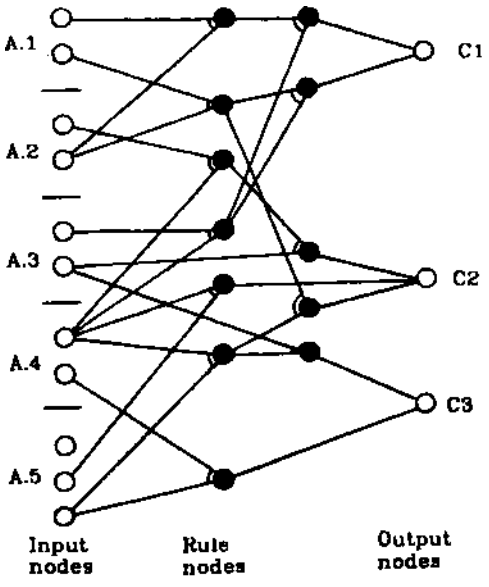

Figure 1

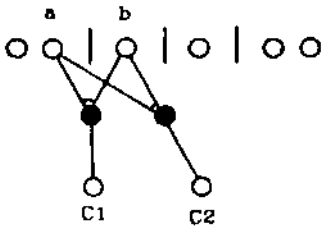

Figure 2

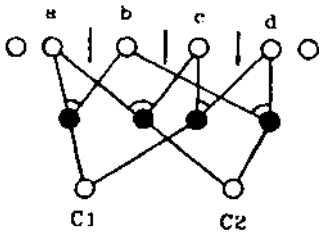

Figure 3

We consider the interaction between classes CI and C2. One would check interactions between every pair of classes in turn. Of course, the backflooding need be done only once per class rather than being repeated for every pair of classes.

We proceed as follows:

Step J. Backflood from node CI. Mark with a "1" all input nodes reached by backflooding. These are called CI-nodes.

Step 2 Backflood from node C2. Mark with a "2" all input nodes reached by backflooding. These are C2-nodes. The 12-nodes are now determined and printed for reviewing by the expert. It then may be clear by inspection that no meaningful input vector is a subset of the set V_{12} of 12-nodes; one could input V_{12} as a vector (or pseudovector) to doublecheck that no class is identified with this (pseudo)vector. (See Step 3.)

Step 3. If V_{12} is not obviously void of meaningful input vectors, then input V_{12} as a (pseudo)vector. There are three possible outcomes.

(a) V_{12} is neither a C1-vector or a C2-vector.

Go to Step 4.

(b) V_{12} is in one class but not the other.

Go to Step 5.

(c) V_{12} is both a C1-vector and a C2-vector.

Go to Step 6.

Step 4. There is no ambiguity between classes C1 and C2.

Step 5 Suppose V_{12} is classified CI but not C2. V_{12} could contain a CI-vector which is part of a C2-vector not contained within V_{12}; thus the minimal CI-vectors within V_{12} should be isolated so this can be checked. This can be done either by the Critical Set algorithm (see below) or by explicit testing. Space limitation prevents further details. The symmetric case of V_{12} classified C2 but not CI is processed similarly.

Step 6. Since both classes CI and C2 are reached by V_{12}, we need to check for meaningful ambiguous vectors within V_{12}. If V_{12} is small then one reasonable way is to input directly the atomic meaningful vectors within V_{12}, checking that they each are uniquely classified. If some are not, one can directly print out the inference steps of the ambiguous rule or seek min-ambiguous subvectors. This can be done bottom-up from singleton nodes or top-down via the Critical Set algorithm (see below). The advantage of finding min-ambiguous subvectors is to ease the Inspection of the inference steps since one knows the critical input nodes yielding the ambiguity. Finally, if too many meaningful subvectors of V_{12} exist to process them directly the Critical Set algorithm may be used.

This ends the procedure description.

We briefly outline the Critical Set algorithm purpose and form and then give a small example.

Space prohibits a full presentation of the Critical Set algorithm, which is presented in [Loveland, 1982]. The algorithm is -a divide-and-conquer algorithm to solve the following problem. We are given a universe U whose power set is the domain of a monotonic set function f, f binary valued, with f( )=O, f(U)=I and ScT => f(S) <. f(T). With f so defined, there exist *critical sets* B such that f(B)=I but for every proper subset S of B, f(S)=0. Thus, B is a "minimal I-set" under f. In Step 5 we use this algorithm where the universe U is the set of 12-nodes and f is defined as f(S)=I if S as a pseudovector yields class CI, f(S)=0 if S yields no class at all. In Step 6, the universe is the same and f(S)=I if S as a pseudovector yields classes CI and C2, f(S)=0 if S is not ambiguously classified. Note that if there are no inhibitor rules then f is monotonic in both cases since the superposition property holds.

The algorithm finds some one critical set, element by element. To find an element it splits successive sets in two until a singleton set is reached, whereupon the isolated element is added to the critical set. To split a pseudovector, divide the values of one attribute in half, add a different half to each set S_1 and S_2 and add all other nodes (from other attributes) to both S_1 and S_2- To split a legal vector (one node per attribute) simply split the nodes evenly between S_1 and S_2-

The algorithm is very efficient (in a suitable sense) in finding some critical set, but finding many critical sets can occasionally lead to high computational cost. For a full discussion see [Loveland, 1982].

We conclude with a very brief illustration regarding the process of finding unintended ambiguities.

Suppose a,b,c,d,e are attribute-value pairs with (only) a and b representing different values for the same attribute. Suppose vectors <a c d>eCI, <b c>eC2, and <c d e>eC2 are the only meaningful vectors. Then V_{12} = (c.d}, learned from backflooding. <c d> is not of either class. <a c d e> is an ambiguous vector since it is in both classes, for good reason as it is a superset of two distinct and differently classed vectors. It is both meaningful and a min-ambiguous vector. <a b c d> is an ambiguous pseudovector, but not a vector since a and b have the same attribute.

Suppose in addition that <b c d>eCI is created in error by a new rule. Then V_{12} = [b.c.d}, and <b c d> is min-ambiguous as can be found by the Critical Set algorithm applied at Step 6. The algorithm gives a suggested sequence of input pseudovectors to be submitted by the expert (or perhaps automatically) to determine the min-ambiguous vector. The algorithm would have one evaluate input vectors <b c>, <d>, <b d> and <c d> (and we know the <b c d> classification as Step 6 was reached). Here the expert on his own might have tried <b c> first, then <c d>, and <b d>, and, getting the expected results, realized the same conclusion slightly more efficiently. We stress that we do not want to oversell the Critical Set algorithm; it is the general approach we emphasize. Most important, we stress the value of testing procedures to catch violations of key properties of dynamic inference systems.

References

[Davis, 1982] Davis, R. "TEIRESIAS: Applications of meta-level knowledge." Part 2 of *Expert Knowledge Systems in Artificial Intelligence,* by Randall Davis and Douglas B Lenat. McGraw-Hill. 1982.

[Loveland, 1982] Loveland, D.W. "Finding critical sets." C.S. Report CS-1982-23, Department of Computer Science, Duke University. Durham. NC, 1982.

[Michalski and Chilauski, 1980] Michalski, R.T. and R.L. Chilauski. "An experimental comparison of the two methods of knowledge acquisition in the context of developing an expert system for soybean disease diagnosis." *fnternational Journal of Policy Analysis and Information Systems, 4,* 2(1980), 125-161.

[Suwa, Scott, and Shortliffe, 1982] Suwa, M., A.C. Scott, and E.H. Shortliffe. "An approach to verifying completeness and consistency in a rule-based expert system." *AI Magazine 3,* Fall 1982, 16-21.