On Debugging Rule Sets When Reasoning Under Uncertainty

David C. Wilkins and Bruce G. Buchanan

Department of Computer Science
Stanford University
Stanford, CA 94305

ABSTRACT

Heuristic inference rules with a measure of strength less than certainty have an unusual property: better individual rules do not necessarily lead to a better overall rule set. All less-than-certain rules contribute evidence towards erroneous conclusions for some problem instances, and the distribution of these erroneous conclusions over the instances is not necessarily related to individual rule quality. This has important consequences for automatic machine learning of rules, since rule selection is usually based on measures of quality of individual rules.

In this paper, we explain why the most obvious and intuitively reasonable solution to this problem, incremental modification and deletion of rules responsible for wrong conclusions *a la* Teiresias, is not always appropriate. In our experience, it usually fails to converge to an optimal set of rules. Given a set of heuristic rules, we explain why the the best rule set should be considered to be the element of the power set of rules that yields a global minimum error with respect to generating erroneous positive and negative conclusions. This selection process is modeled as a bipartite graph minimization problem and shown to be NP-complete. A solution method is described, the Antidote Algorithm, that performs a model-directed search of the rule space. On an example from medical diagnosis, the Antidote Algorithm significantly reduced the number of misdiagnoses when applied to a rule set generated from 104 training instances.

I Introduction

Reasoning under uncertainty has been widely investigated in artificial intelligence. Probabilistic approaches are of particular relevance to rule-based expert systems, where one is interested in modeling the heuristic and evidential reasoning of experts. Methods developed to represent and draw inferences under uncertainty include the certainty factors used in Mycin [2], fuzzy set theory [12], and the belief functions of Dempster-Shafer theory [10] [5]. In many expert system frameworks, such as Emycin, Expert, MRS, S.1, and Kee, the rule structure permits a conclusion to be drawn with varying degrees of certainty or belief. This paper addresses a concern common to all these methods and systems.

In refining and debugging a probabilistic rule set, there are three major causes of errors: *missing* rules, *wrong* rules, and *deleterious interactions* between good rules. The purpose of this paper is to explicate a type of deleterious interaction and to show that it (a) is indigenous to rule sets for reasoning under uncertainty, (b) is of a fundamentally different nature from missing and wrong rules, (c) cannot be handled by traditional methods for correcting wrong and missing rules, and (d) can be handled by the method described in this paper.

In section 2, we describe the type of deleterious rule interactions that we have encountered in connection with automatic induction of rule sets, and explain why the use of most *rule modification methods* fails to grasp the nature of the problem. In section 3, we discuss approaches to debugging and refining rule sets and explain why traditional *rule set debugging methods* are inadequate for handling global interactions. In section 4, we formulate the problem of reducing deleterious interactions as a bipartite graph minimization problem and show that it is NP-complete. In section 5, we present a heuristic solution method called the Antidote Algorithm. Finally, our experiences in using the Antidote Algorithm are described.

A brief description of terminology will be helpful to the reader. Assume there exists a collection of *training instances*, each represented as a set of feature-value pairs of *evidence* and a set of *hypotheses*. Rules have the form LHS $\longrightarrow$ RHS (CF) , where LHS is a conjunction of evidence, RHS is a hypothesis, and CF is a certainty factor or its equivalent. A rule that correctly confirms a hypothesis generates *true positive* evidence; one that correctly disconfirms a hypothesis generates *true negative* evidence. A rule that incorrectly confirms a hypothesis generates *false positive* evidence; one that incorrectly disconfirms a hypothesis generates *false negative* evidence. False positive and false negative evidence can lead to *misdiagnoses* of training instances.

II Inexact Reasoning and Rule Interactions

When operating as an evidence-gathering system [2], an expert system accumulates evidence for and against competing hypotheses. Each rule whose preconditions match the gathered data contributes either positively or negatively toward one or more hypotheses. Unavoidably, the preconditions of probabilistic rules succeed on instances where the rule will be contributing false positive or false

negative evidence for conclusions. For example, consider the following rule:[1]

R1: Surgery=Yes ∧ Gram_Neg_Infection=Yes

⇒ Klebsiella=Yes (0.77)

The frequency with which R1 generates false positive evidence has a major influence on its CF of 0.77, where $-1 \leq \text{CF} \leq 1$. Indeed, given a set of training instances, such as a library of medical cases, the certainty factor of a rule can be given a probabilistic interpretation[2] as a function $\Phi(x_1, x_2, x_3)$, where x_1 is the fraction of the positive instances of a hypothesis where the rule premise succeeds, thus contributing true positive or false negative evidence; x_2 is the fraction of the negative instances of a hypothesis where the rule premise succeeds, thus contributing false positive or true negative evidence; and x_3 is the ratio of positive instances of a hypothesis to all instances in the training set. For R1 in our domain, $\Phi(.43, .10, .22) = 0.77$, because statistics on 104 training instances yield the following values:

x_1: LHS true among positive instances = 10/23
x_2: LHS true among negative instances = 8/81
x_3: RHS true among all instances = 23/104

Hence, R1 generates false positive evidence on eight instances, some of which may lead to false negative diagnoses. But whether they do or not depends on the other rules in the system; hence our emphasis on taking a global perspective. The usual method of dealing with situations such as this is to make the rule fail less often by specializing its premise [8]. For example, surgery could be specialized to neurosurgery, and we could replace R1 with:

R2: *Neuro*surgery=Yes ∧ Gram_Neg_Infection=Yes

⇒ Klebsiella=Yes (0.92)

On our case library of training instances for the R2 rule, $\Phi(.26, .02, .22) = 0.92$, so R2 makes erroneous inferences in two instances instead of eight. Nevertheless, modifying R1 to be R2 on the grounds that R1 contributes to a misdiagnosis is not always appropriate; we offer three objections to this frequent practice. First, both rules are *inexact* rules that offer advice in the face of limited information, and their relative accuracy and correctness is explicitly represented by their respective CFs. We expect them to fail, hence failure should not necessarily lead to their modification. Second, all probabilistic rules reflect a trade-off between generality and specificity. An overly general rule provides too little discriminatory power, and a overly specific rule contributes too infrequently to problem solving. A policy on proper grain size is explicitly or implicitly built into rule induction programs; this policy should be followed as much as possible. Specialization produces a rule that usually violates such a policy. Third, if the underlying problem for an incorrect diagnosis is rule interactions, a more specialized rule, such as the specialization of R1 to R2, can be viewed as creating a potentially more dangerous rule. Although it only makes an incorrect inference in two instead of eight instances, these two instances will be now harder to counteract when they contribute to misdiagnoses because R2 is stronger. Note that a rule with a large CF is more likely to have its erroneous conclusions lead to misdiagnoses. This perspective motivates the prevention of misdiagnoses in ways other than the use of rule specialization or generalization.

Besides rule modification, another way of nullifying the incorrect inference of a rule in an evidence-gathering system is to introduce a counteracting rules. In our example, this would be rules with a negative CF that concludes Klebsiella on the false positive training instances that lead to misdiagnoses. But since these new rules are probabilistic, they introduce false negatives on some other training instances, and these may lead to misdiagnoses. We could add yet more counteracting rules with a positive CF to nullify any problems caused by the original counteracting rules, but these rules introduce false positives on yet other training instances, and these may lead to other misdiagnoses. Also, a counteracting rule is often of less quality in comparison to rules in the original rule set; if it were otherwise the induction program would have included the counteracting rule in the original rule set. Clearly, adding counteracting rules may not be necessarily the best way of dealing with misdiagnoses made by probabilistic rules.

III Debugging Rule Sets and Rule Interactions

Assume we are given a set of probabilistic rules that were either automatically induced from a set of training cases or created manually by an expert and knowledge engineer. In refining and debugging this probabilistic rule set, there are three major causes of errors: missing rules, wrong rules, and unexpected interactions among good rules. We first describe types of rule interactions, and then show how the traditional approach to debugging is inadequate.

A. Types of rule interactions

In a rule-based system, there are many types of rule interactions. Rules interact by *chaining* together, by using *the same evidence* for different conclusions, and by drawing the *same conclusions* from different collections of evidence. Thus one of the lessons learned from research on MYCIN [2] was that complete modularity of rules is not possible to achieve when rules are written manually. An expert uses other rules in a set of closely interacting rules in order to define a new rule, in particular to set a CF value relative to the CFs of interacting rules.

[1] This is a simplified form of ($And (Same Cntxt Surgery)) ⇒ (Conclude Cntxt Gram_Negative_1 Klebsiella Tally 770).

[2] See Appendix 1 for a description of the function Φ. This statistical interpretation of CFs deemphasizes incorporating orthogonal utility measures as discussed in [2].

Automatic rule induction systems encounter the same problems. Moreover, automatic systems lack an understanding of the strong semantic relationships among concepts to allow judgements about the relative strengths of evidential support. Instead, induction systems use *biases* to guide the rule search [8] [3]. Examples of some biases used by the induction subsystem of the Odysseus apprenticeship learning program are rule generality, whereby a rule must cover a certain percentage of instances; rule specificity, whereby a rule must be above a minimum discrimination threshold; rule colinearity, whereby rules must not be too similar in classification of the instances in the training set; and rule simplicity, whereby a maximum bound is placed on the number of conjunctions and disjunctions [3].

B. Traditional methods of debugging a rule set

The standard approach to debugging a rule set consists of iteratively performing the following steps:

- Step 1. Run the system on cases until a false diagnosis is made.
- Step 2. Track down the error and correct it, using one of five methods pioneered by Teiresias [4] and used by knowledge engineers generally:
  - Method 1: Make the preconditions of the offending rules more specific or sometimes more general.[3]
  - Method 2: Make the conclusions of offending rules more general or sometimes more specific.
  - Method 3: Delete offending rules.
  - Method 4: Add new rules that counteract the effects of offending rules.
  - Method 5: Modify the strengths or CFs of offending rules.

This approach may be sufficient for correcting wrong and missing rules. However, it is flawed from a theoretical point of view, with respect to its sufficiency for correcting problems resulting from the global behavior of rules over a set of cases. It possesses two serious methodological problems. First, using all five of these methods is not necessarily appropriate for dealing with global deleterious interactions. In section 2 we explained why in some situations modifying the offending rule or adding counteracting rules leads to problems, and misses the point of having probabilistic rules, and this eliminates methods 1, 2 and 4. If rules are being induced from a training set of cases, modifying the strength of the rule is illegal, since the strength of the rule has a probabilistic interpretation, being derived from frequency information derived from the training instances, and this eliminates method 5. Only method 3 is left to cope with deleterious interactions. The second methodological problem is that the traditional method picks an arbitrary case to run in its search for misdiagnoses. Such a procedure will often not converge to a good rule set, even if modifications are restricted to rule deletion. Example 2 in section 5.B illustrates this situation.

Our perspective on this topic evolved in the course of experiments in induction and refinement of knowledge bases. Using "better" induction biases did not always produce rule sets with better performance, and this prompted investigating the possibility of global probabilistic interactions. Our original approach to debugging was similar to the Teiresias approach. Often, correcting a problem led to other cases being misdiagnosed, and in fact this type of *automated* incremental debugging seldom converged to an acceptable set of rules. It might have if we we engaged in the common practice of "tweaking" the CF strengths of rules. However this was not permissible, since our CF values have a precise probabilistic interpretation.

IV Problem Formalization

Assume there exists a large set of training instances, and a rule set for solving these instances has been induced that is fairly complete and contains rules that are individually judged to be good. By good, we mean that they individually meet some predefined quality standards such as the biases described in section 3.A. Further, assume that the rule set misdiagnoses some of the instances in the training set. Given such an initial rule set, the problem is to find a rule set that meets some optimality criteria, such as to minimize the number of misdiagnoses without violating the goodness constraints on individual rules.[4] Now modifications to rules, except for rule deletion, generally break the predefined goodness constraints. And adding other rules is not desirable, for if they satisfied the goodness constraints they would have been in the original rule set produced by the induction program. Hence, if we are to find a solution that meets the described constraints, the solution must be a subset of the original rule set.[5]

The best rule set is viewed as the element of the power set of rules in the initial rule set that yields a global minimum weighted error. A straightforward approach is to examine and compare all subsets of the rule set. However, the power set is almost always to large to work with, especially when the initial set has deliberately been generously generated. The selection process can be modeled as a bipartite graph minimization problem as follows.

[3] Ways of generalizing and specializing rules are nicely described in [8]. They include dropping conditions, changing constants to variables, generalizing by internal disjunction, tree climbing, interval closing, exception introduction, etc.

[4] In Meta-Dendral, a large initial rule set was created by the RULEGEN program, which produced plausible individual rules without regard to how the rules worked together. The RULEMOD program selected and refined a subset of the rules. See [1] for details.

[5] If we discover that this solution is inadequate for our needs, then introducing rules that violate the induction biases is justifiable.

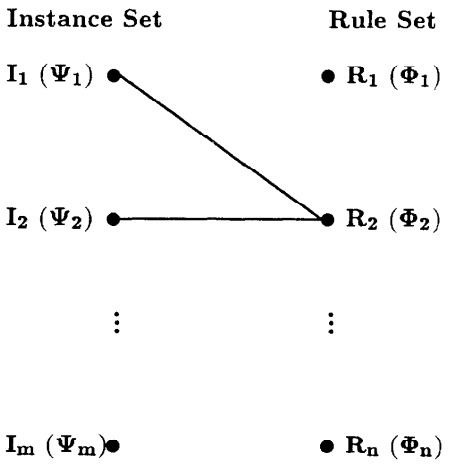

Figure 1: Bipartite Graph Formulation

A. Bipartite graph minimization formulation

For each hypothesis in the set of training instances, define a directed graph $G(V, A)$, with its vertices V partitioned into two sets I and R, as shown in Figure 1. Elements of R represent rules, and the evidential strength of R_j is denoted by Φ_j. Each vertex in I represents a training instance; for positive instances Ψ_i is 1, and for negative instances Ψ_i is -1. Arcs $[R_j, I_i]$ connect a rule in R with the training instances in I for which its preconditions are satisfied; the weight of arc $[R_j, I_i]$ is Φ_j. The weighted arcs terminating in a vertex in I are combined using an evidence combination function Φ', which is defined by the user. The combined evidence classifies an instance as a positive instance if the combined evidence is above a user specified threshold $\mathrm{CF_t}$. In the example in section V.B., CF_t is 0, while for Mycin, $\mathrm{CF_t}$ is 0.2.

More formally, assume that $I_1, \ldots, I_m$ = training set of instances, and $R_1, \ldots, R_n$ = rules of an initial rule set. Then we want to minimize:

$$z = \sum_{j=1}^{n} b_j r_j$$

subject to the constraints

$$\bigwedge_{i=1}^{m} \left(\Phi'(a_{i1} r_1, \ldots, a_{in} r_n) \otimes_i \mathrm{CF_t} \right)$$

$$\sum_{j=1}^{n} r_j \geq R_{min}$$

where

$r_j =$ if R_j is in solution rule set then 1 else 0;

$b_j =$ bias constant to preferentially favor rules;

$a_{ij} =$ if arc $[R_j, I_i]$ exists then Φ_j else 0;

$CF_t =$ the CF threshold for positive classification;

$\Phi' =$ n-ary function for combining CFs, where the time to evaluate is polynomial in n;

$R_{min} =$ minimum number of rules in solution set;

$\otimes_i =$ if Ψ_i is 1 then " $>$ " else " $\leq$ ".

The solution formulation solves for r_j; if $r_j = 1$ then rule R_j is in the final rule set. The main task of the user is setting up the a_{ij} matrix, which associates rules and instances and indicates the strength of the the associations. Note that the value of a_{ij} is zero if the preconditions of R_j are not satisfied in instance I_i Preference can be given to particular rules via the bias b_j in the objective function z. For instance, the user may wish to favor the selection of strong rules. The R_{min} constraint forces the solution rule set to be above a minimum size. This prevents finding a solution that is too specialized for the training set, giving good accuracy on the training set but having a high variance on other sets, which would lead to poor performance.

Theorem 1. The bipartite graph minimization problem for heuristic rule set optimization is NP-complete.

Proof. A sketch of our proof is given; details can be found in [11]. To show that the bipartite graph minimization problem is NP-complete, we use reduction from Satisfiability. Satisfiability clauses are mapped into graph instance nodes and the atoms of the clauses are mapped into rule nodes. Arcs connect rule nodes to instance nodes when the respective literals appear in the respective clauses. The evidence combination function ensures that at least one arc goes into each clause node from a rule node representing a true literal. The evidence combination function also performs bookkeeping functions. ◇

V Solution Method

In this section, a solution method called the Antidote Algorithm is described, and an example is provided based on the graph shown in figure 2. An alternative solution method that uses zero-one integer programming is described in [11]. It is more robust, but places a restriction on the evidence combination function, namely that the evidence be additively combined. It is not adequate when using the certainty factor model, but may be suitable for connectionist approaches.

A. The Antidote Algorithm

The following model-directed search method, the Antidote Algorithm, is one that we have developed and used in our experiments:

- Step 1. Assign values to penalty constants. Let p_1 be the penalty assigned to a poison rule. A *poison rule* is a strong rule giving erroneous evidence for a case

that cannot be counteracted by the combined weight of all the rules that give correct evidence. Let p_2 be the penalty for contributing false positive evidence to a misdiagnosed case, p_3 be the penalty for contributing false negative evidence to a misdiagnosed case, p_4 be the penalty for contributing false positive evidence to a correctly diagnosed case, p_5 be the penalty for contributing false negative evidence to a correctly diagnosed case, and p_6 be the penalty for using weak rules. Let h be the maximum number of rules that are removed at each iteration. Let R_{min} be the minimum size of the solution rule set.

- Step 2. Optional step for very large rule sets: given an initial rule set, create a new rule set containing the n strongest rules for each case.
- Step 3. Find all misdiagnosed cases for the rule set. Then collect and rank the rules that contribute evidence toward these erroneous diagnoses. The rank of rule R_j is $\sum_{i=1}^{6} p_i n_{ij}$, where:
  - $n_{1i} = 1$ if R_j is a poison rule or its deletion leads to the creation of another poison rule and 0 otherwise.
  - n_{2j} = the number of misdiagnoses for which R_j gives false positive evidence;
  - n_{3j} = the number of misdiagnoses for which R_j gives false negative evidence;
  - n_{4j} = the number of correct diagnoses for which R_j gives false positive evidence;
  - n_{5j} = the number of correct diagnoses for which R_j gives false negative evidence;
  - n_{6j} = the absolute value of the CF of R_j;
- Step 4. Eliminate the h highest ranking rules.
- Step 5. If the number of misdiagnoses begins to increase and $h \neq 1$, then $h \leftarrow h - 1$. Repeat steps 3-4 until either
  - there are no misdiagnoses
  - R_{min} is reached
  - $h = 1$ and the number of misdiagnoses begins to increase. ◇

Each iteration of the algorithm produces a new rule set, and each rule set must be rerun on all training instances to locate the new set of misdiagnosed instances. If this is particularly difficult to do, the h parameter in step 4 can be increased, but there is the potential risk of converging to a suboptimal solution. For each misdiagnosed instance, the automated reasoning system that uses the rule set must be able to explain which rules contributed to a misdiagnosis. Hence, we require an system with good explanation capabilities.

The nature of an optimal rule set differs between domains. Penalty constants, p_i, are the means by which the user can define an optimal policy. For instance, via p_2 and p_3, the user can favor false positive over false negative misdiagnoses, or visa versa. For medical expert systems, a false negative is often more damaging than a false positive, as false positives generated by a medical program can often be caught by a physician upon further testing. False negatives, however, may be sent home, never to be seen again.

In our experiments, the value of the six penalty constants was $p_i = 10^{6-i}$. The h constant determines how many rules are removed on each iteration, with lower values, especially $h \leq 3$, giving better performance. R_{min} is the minimum size of the solution rule set; its usefulness was described in section 5.A.

Example 1.

In this example, which is illustrated in Figure 2, there are six training instances, classified as positive or negative instances of the hypothesis. There are five rules shown with their CF strength. The arcs indicate the instances to which the rules apply. To simplify the example, define the combined evidence for an instance as the sum of the evidence contributed by all applicable rules, and let $CF_t = 0$. Rules with a CF of one sign that are connected to an instance of the other sign contribute erroneous evidence. Two cases in the example are misdiagnosed: I_4 and I_5. The objective is to find a subset of the rule set that minimizes the number of misdiagnoses.

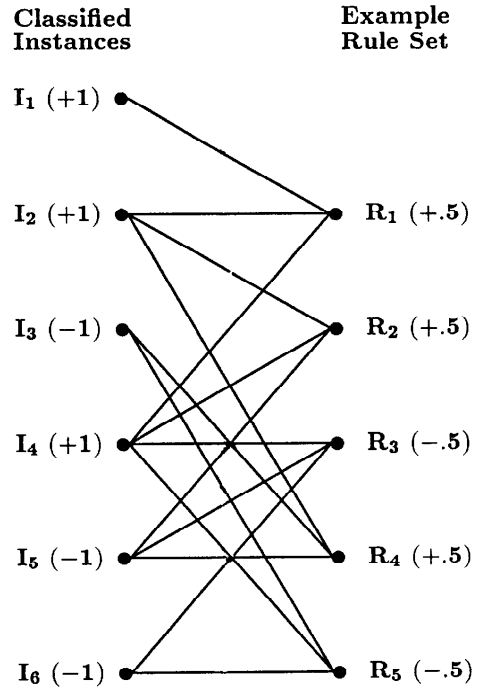

Figure 2: Optimizing Rules for One Hypothesis

Assume that the final ruleset must have at least three rules, hence $R_{min} = 3$. Since all rules have identical magnitude and out degree, it is reasonable to set the bias to the same value for all n rules. hence $b_j = 1$, for $1 \leq j \leq n$.

Let $p_i = 10^{6-i}$, for $0 \leq i \leq 5$, thus choosing rules in the highest category, and using lower categories to break ties.

On the first iteration, two misdiagnosed instances are found, I_4 and I_5, and four rules contribute erroneous evidence toward these misdiagnoses, R_2, R_3, R_4, and R_5. Rules are ranked and R_4 is chosen for deletion. On the second iteration, one misdiagnosis is found, I_4, and two erroneous rules contribute erroneous evidence, R_3 and R_5. Rules are ranked and R_5 is deleted. This reduces the number of misdiagnoses to zero and the algorithm successfully terminates.

The same example can be used to illustrate the problem of the traditional method of rule set debugging, where the order in which cases are checked for misdiagnoses influences which rules are deleted. Consider a Teiresias style program that looks at training instances and discovers I_4 is misdiagnosed. There are two rules that contribute erroneous evidence to this misdiagnosis, rules R_3 and R_5. It wisely notices that deleting R_5 causes I_3 to become misdiagnosed, hence increasing the number of misdiagnoses; so it chooses to delete R_3. However, no matter which rule it now deletes, there will always be at least one misdiagnosed case. To its credit, it reduced the number of misdiagnoses from two to one; however, it fails to converge to an rule set that minimizes the number of misdiagnoses. $\diamondsuit$

B. Experience with the Antidote Algorithm

Experiments with the Antidote Algorithm were performed using the Mycin case library[2]. Our experiments involved using 119 evidential findings, 26 intermediate hypotheses, and 21 final hypotheses. The training set had 104 training instances and each instance was classified as a member of four hypothesis classes on the average. The generated rules had one to three LHS conjuncts.

In our experiments, we generated approximately forty rule sets containing between 200 and 20000 rules. Large rule sets were generated because we our investigating the construction of knowledge bases that allow an expert system to automatically follow the line of reasoning of an expert; understanding a community of problem solvers requires more knowledge than that needed to just solve diagnosis problems. Typically, 85% of the training instances were diagnosed correctly, and seven out of ten cases used to validate the original Mycin system were evaluated correctly. While ten cases is a small number for a validation set, it is a carefully constructed set and has been found adequate in accurately classifying human diagnosticians at all levels [7]. Further, since there are an average of four hypotheses in the diagnosis per instance, we can view our training set as having 416 instances and our validation set as having 40 instances. After, the Antidote Algorithm was applied, 95% of the training instances was diagnosed correctly, and 80% of the validation set was diagnosed correctly.

Besides almost always converging to a solution in which all members of the training set are diagnosed correctly, the Antidote Algorithm is very efficient: only five to fifteen iterations are required, for rule sets containing between 200 and 500 rules. It was surprising to see how greatly performance is improved by deleting a small percentage of the rules in the rule set. As our results show, the improved performance on the training set carried over to the validation set.

VI Summary and Conclusion

Traditional methods of debugging a probabilistic rule set are suited to handling missing or wrong rules, but not to handling deleterious interactions between good rules. This paper describes the underlying reason for this phenomenon. We formulated the problem of minimizing deleterious rule interactions as a bipartite graph minimization problem and proved that it is NP-Complete. A heuristic method was described for solving the graph problem, called the Antidote Algorithm. In our experiments, the Antidote Algorithm gave good results. It reduced the number of misdiagnoses on the training set from 15% to 5%, and the number of misdiagnoses on the validation set from 30% to 20%.

We believe that the rule set refinement method described in this paper, or its equivalent, is an important component of any learning system for automatic creation of probabilistic rule sets for automated reasoning systems. All such learning systems will confront the problem of deleterious interactions among good rules, and the problem will require a global solution method, such as we have described here.

VII Acknowledgements

We thank Marianne Winslett for suggesting the bipartite graph formulation and for detailed comments. We also express our gratitude for the helpful discussions and critiques provided by Bill Clancey, Ramsey Haddad, David Heckerman, Eric Horovitz, Curt Langlotz, Peter Rathmann and Devika Subramanian.

This work was supported in part by NSF grant MCS-83-12148, ONR/ARI contract N00014-79C-0302, Advanced Research Project Agency Contract DARPA N00039-83-C-0136, the National Institute of Health Grant NIH RR-00785-11, National Aeronautics and Space Administration Grant NAG-5-261, and Boeing Grant W266875. We are grateful for the computer time provided by the Intelligent Systems Lab of Xerox PARC and SUMEX-AIM.

Appendix 1: Calculating Φ.

Consider rules of the form E $\xrightarrow{CF}$ H. Then CF = Φ = $\Phi(x_1, x_2, x_3)$ = empirical predictive power of rule R, where:

- $x_1 = P(E^+|H^+)$ = fraction of the positive instances in which R correctly succeeds (true positives or true negatives)

- $x_2 = P(E^+|H^-)$ = fraction of the negative instances in which R incorrectly succeeds false positives or negatives
- $x_3 = P(H^+)$ = fraction of all instances that are positive instances

Given x_1, x_2, x_3, let

- $x_4 = P(H^+|E^+) = \frac{x_1 x_3}{x_1 x_3 + x_2(1-x_3)}$.

If $x_4 > x_3$ then $\Phi = \frac{x_4 - x_3}{x_4(1-x_3)}$ else $\Phi = \frac{x_4 - x_3}{x_3(1-x_4)}$.

This probabilistic interpretation reflects to the modifications to the certainly factor model proposed by [6].

REFERENCES

[1] B. G. Buchanan and T. M. Mitchell. Model-directed learning of production rules. In *Pattern-Directed Inference Systems*, pages 297–312, New York: Academic Press, 1978.

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

[3] Wilkins D. C., Clancey W. J., and B. G. Buchanan. *An overview of the Odysseus learning apprentice*, pages 332–340. New York: Kluwer Academic Press, 1986.

[4] R. Davis and D. B. Lenat. *Knowledge-Based Systems in Artificial Intelligence.* McGraw-Hill, New York, 1982.

[5] J. Gordon and E. H. Shortliffe. A method for managing evidential reasoning in a hierarchical hypothesis space. *Artificial Intelligence*, 26(3):323–358, July 1985.

[6] D. Heckerman. Probabilistic interpretations for Mycin's certainty factors. In *Uncertainty in Artificial Intelligence*, North Holland, 1986.

[7] Yu V. L., Fagan L. M., et al. Evaluating the performance of a computer-based consultant. *J. Amer. Med. Assoc.*, 242(12):1279–1282, 1979.

[8] R. S. Michalski. *A theory and methodology of inductive inference*, chapter 4, pages 83–134. Palo Alto: Tioga, 1984.

[9] P. Politakis and S. M. Weiss. Using empirical analysis to refine expert system knowledge bases. *Artificial Intelligence*, 22(1):23–48, 1984.

[10] G. A. Shafer. *Mathematical Theory of Evidence.* Princeton University Press, Princeton, 1976.

[11] D. C. Wilkins and B. G. Buchanan. *On debugging rule sets when reasoning under uncertainty.* Technical Report KSL 86-30, Stanford University, Computer Science Dept., 1986.

[12] L. A. Zadeh. Approximate reasoning based on fuzzy logic. In *IJCAI-6*, pages 1004–1010, 1979.